BAB III

METODOLOGI PENELITIAN DAN PERANCANGAN SISTEM

3.1 Metodologi Penelitian

Dalam penelitian ini, metodologi penelitian yang dilakukan adalah sebagai berikut.

1. Studi Literatur

   Pembelajaran teori, konsep, dan algoritma yang berkaitan dengan penelitian. Teori yang dipelajari adalah *boxing*, *human-computer interaction*, *game*, *game design*, Unity, Kinect, *gesture recognition*, algoritma Fisher-Yates Durstenfeld, dan *Likert Scale*.

2. Analisis Kebutuhan

   Analisis kebutuhan fitur-fitur yang akan dibutuhkan untuk melakukan perancangan dan pembangunan *game* 3D seperti kebutuhan controller apa saja yang dibuat. *Controller* merupakan objek yang mengatur alur sistem yang ada pada *game*. Terdapat banyak jenis *controller* sebagai contoh *controller* yang mengatur gerakan *player* dan *enemy* dan *controller* yang mengatur *scoring* pada *game*.

3. Perancangan Aplikasi

   Perancangan *flowchart* dibuat untuk memudahkan melihat alur keseluruhan dari sistem *game*. Setelah perancangan selesai, dilanjutkan dengan perancangan *player interface*. *Player interface* yang dirancang akan *user friendly* dan dibuat agar cocok untuk berinteraksi dengan menggunakan

sensor Kinect. Setelah itu dilakukan proses pembuatan *gestures* yang dibutuhkan dalam *game* dengan menggunakan Kinect Studio dan Visual Gesture Builder Preview.

4. Pembangunan Aplikasi

   Pembangunan aplikasi dilakukan dengan Unity versi 5.3.3f1 64 bit menggunakan C# sebagai bahasa pemogramannya. Kinect MS-SDK versi 2 digunakan untuk menghubungkan aplikasi dengan sensor Kinect versi 2. Visual Gesture Builder merupakan fitur terbaru Kinect versi 2 yang akan digunakan untuk membuat *recognition gesture boxing* untuk tiap-tiap *gesture* yang dibuat dengan menggunakan Kinect Studio dan Visual Gesture Builder Preview. Perancangan *player interface* akan dibuat dengan menggunakan Corel Draw X5.

5. *Testing* dan Survei

   Tahap pengujian aplikasi *game* dilakukan dengan Asus Notebook X550C dan Kinect versi 2. Proses pengujian akan dilakukan oleh partisipan. Setelah aplikasi selesai dimainkan, partisipan akan diberikan kuesioner yang digunakan sebagai evaluasi aplikasi. Jumlah peserta yang akan dievaluasi adalah 30 orang atau lebih. Menurut Gay dan Diehl (1992), 30 *sample* merupakan jumlah minimum *sample* yang diperlukan dalam melakukan suatu penelitian.

6. Evaluasi

   Evaluasi akan berisi evaluasi dengan analisa berdasarkan data yang telah dikumpulkan dari pengisian kuesioner setelah para partisipan memainkan *game*. Likert Scale merupakan metode penyimpulan aplikasi yang akan

digunakan untuk mengetahui persentase keberhasilan aplikasi dalam memperkenalkan *boxing* dan evaluasi penilaian rancangan *player interface*.

3.2 Struktur Game

Judul *game* : Boxing Action

Formal elements yang terdapat pada *game* adalah sebagai berikut.

1. Player: *Single player game*
2. Objectives: Mengalahkan musuh dengan cara menghabiskan *health point*-nya terlebih dahulu daripada *health point player*. Untuk mengurangi *health point enemy* harus mengingat gerakan pertahanan musuh terlebih dahulu lalu melancarkan serangan yang tidak bisa di-*block* gerakan pertahanan musuh dan kemudian *player* harus bertahan dari serangan musuh dengan cara mengingat gerakan serangan musuh lalu mem-*block* dengan gerakan pertahanan yang tepat untuk gerakan serangan.
3. Procedures
    1. Terdapat 4 pilihan *menu battle, training, credit, leaderboard*. Untuk berpindah ke *menu* lain lakukan *swipe right* atau *swipe left gesture*.
    2. Pada menu *battle player* akan melawan musuh yang gerakannya dibangkitkan dengan menggunakan algoritma *shuffle*, ingat semua gerakan musuh dan balas dengan gerakan yang tepat untuk mendapatkan *point right answer* dan mengalahkan musuh. Jika salah dalam memilih gerakan, *player* akan kehilangan *health point* dan mendapatkan *point wrong answer*.
    3. Pada menu *training player* dapat mempelajari gerakan-gerakan serangan dan pertahanan dasar dari *boxing*.

4. Pada menu *credit* akan ditampilkan orang-orang yang bersangkutan dalam pembangunan *game*.
5. Pada menu *leaderboard* akan ditampilkan 5 skor terbaik dari hasil pertarungan *player* dan *enemy*.

4. Rules
    1. *Player* dan *enemy* masing-masing mendapatkan giliran bertahan dan menyerang. Ketika musuh menyerang *player* bertahan dan ketika musuh bertahan *player* menyerang. Untuk melakukan serangan atau pertahanan *player* harus menggerakan bagian tubuh sesuai dengan gesture yang tersedia.
    2. *Game* bertipe *turn-based game* yang berati *player* dan *enemy* bergiliran untuk melakukan sesuatu. *Game* selalu dimulai dengan musuh menyerang *player*.
    3. Setelah semua *input*-an dari *player* selesai dimasukkan *fighting scene* akan ditampilkan.
    4. Jika gerakan *player* kurang dari jumlah total gerakan yang harus dilakukan, sisa gerakan yang tidak diberikan oleh *player* dianggap gerakan yang salah dan akan mengurangi *health point*.
    5. Panel gerakan *enemy* berwarna merah dan panel gerakan *player* biru.
    6. Terdapat 3 *level* di dalam *game*, mudah gerakan musuh berjumlah 2 sampai 3, sedang gerakan musuh berjumlah 4 sampai 6, dan sulit gerakan musuh berjumlah 5 sampai 8 dan terdapat 12 ronde dalam game dan setiap ronde ada 6 giliran untuk player dan enemy. Tiap giliran player dan enemy mendapatkan jumlah gerakan yang sama.

5. Boundaries

    1. *Player* hanya bisa melakukan 8 gerakan serangan dan 5 gerakan pertahanan.
    2. *Game* hanya dapat dimainkan dengan menggunakan sensor Kinect sebagai alat interaksinya.
    3. *Player* harus berada dalam jangkauan sensor Kinect agar dapat dideteksi keberadaannya dan gerakannya.

6. Conflict

    1. Jika gerakan *player* kurang dari jumlah total gerakan yang harus dilakukan, sisa gerakan yang tidak diberikan oleh *player* dianggap gerakan salah dan akan mengurangi *health point*.
    2. Semakin tinggi tingkat kesulitan, maka jumlah gerakan yang harus diingat semakin banyak juga.
    3. Gerakan serangan harus ditangkis dengan gerakan pertahanan yang benar. Jika salah dalam memilih gerakan, akan berakibat kekurangan *health point* pada *player* atau gagal mengurangi *health point enemy* kalau gagal dalam memilih gerakan serangan yang benar.

7. Resources

    1. Ronde, merupakan total jumlah ronde yang ada pada *game*.
    2. *Score*, jumlah skor gerakan benar dan skor gerakan salah.
    3. *Turn*, jumlah giliran di tiap ronde.
    4. Gerakan serangan dan pertahanan yang terdapat di dalam animasi 3D *game*.
    5. *Health point* yang dimiliki oleh *player* dan *enemy*.

8. Outcome

Outcome dari *game* adalah zero-sum. Jika *player* menang, musuh kalah dan jika *player* kalah, musuh menang.

Terdapat 5 *dramatic elements* di dalam *game* sebagai berikut.

1. Challenge
    1. Waktu yang terbatas untuk mengingat gerakan-gerakan musuh.
    2. Jumlah gerakan tiap giliran berbeda-beda, jika kurang meng-input gerakan dari jumlah total gerakan akan dianggap salah.
2. Play

    Turn-based play merupakan *game* yang terdapat giliran antar pemain untuk melakukan sesuatu.
3. Premise

    Bob Drake adalah seorang petinju yang memiliki *the sixth sense* yang bisa melihat gerakan-gerakan apa saja yang akan dilakukan oleh lawan. Akan tetapi hanya bisa melihat sekilas dan ternyata Bob memiliki daya ingat yang sangat lemah dan reaksi yang lamban akibat kecelakaan yang menimpa dirinya saat kecil. Drake pun bertekad melatih daya ingatan dan pergi ke suatu tempat pelatihan *boxing academy* yang ternama yaitu Boxing Action.
4. Character

    Bob Drake, seorang petinju yang memiliki *the sixth sense* yang dapat melihat gerakan lawan yang akan dilancarkan tetapi tidak dapat mengingat gerakan lawan tersebut karena memiliki daya ingat yang lemah.

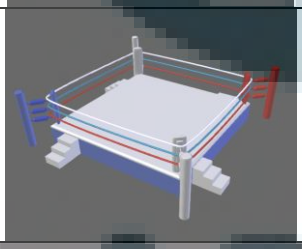
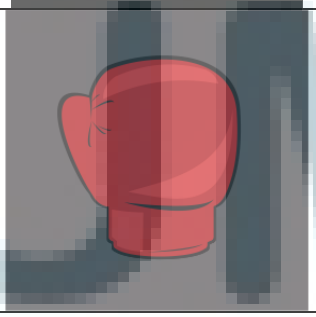
5. Story

Felix Drake dan Emily Drake merupakan ayah dan ibu dari Bob Drake. Felix merupakan petinju yang terkenal dan Emily adalah seorang sejarahwan yang tekenal juga. Bob merupakan anak yang baik yang diberkahi ingatan yang kuat dari ibunya dan saraf motorik yang cepat bereaksi dari ayahnya, anak dari Felix ini bercita-cita ingin menjadi petinju seperti ayahnya, hal ini tentu diberikan dukungan penuh dari orang tua Bob. Bob selalu berlatih dengan ayahnya setiap hari hingga suatu saat Bob menemukan bahwa dirinya mempunyai *the sixth sense* yang membuatnya dapat melihat sekilas beberapa gerakan-gerakan yang akan dilakukan oleh ayahnya saat berlatih. Setelah mengetahui hal itu ayah dan ibu Bob melarangnya untuk memberitahu ke siapapun. Akan tetapi, anak yang berambut pirang ini tidak sengaja memberitahu teman-temannya karena merasa iri teman-teman Bob mengeroyok Bob langsung ditempat dan kepala Bob dibenturkan ke dinding. Saat kejadian itu terjadi orang-orang melihat dan langsung membawa Bob ke rumah sakit. Nasib buruk menimpa anak malang ini, Bob diduga mengalami penurunan daya ingat akibat benturan keras pada kepala dan saraf-saraf motoriknya menjadi lamban karena cedera yang diterimanya. Akan tetapi, hal ini tidak membuat niat Bob Drake menjadi petinju menjadi hilang. Bob Drake terus menerus berlatih untuk memulihkan kembali daya ingat dan saraf motoriknya di tempat yang bernama Boxing Action. Bantulah Bob Drake menjadi juara boxing.

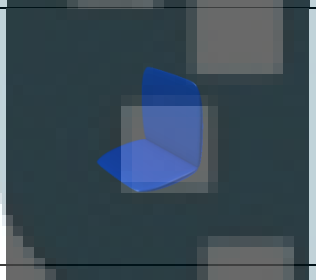
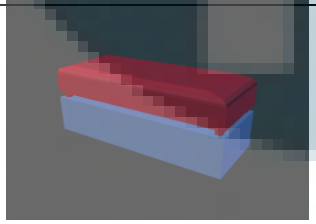
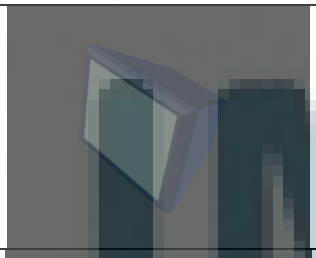
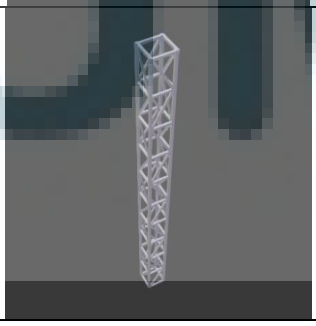
3.3 Penggunaan Asset

Aset-aset yang digunakan pada *game* adalah sebagai berikut.

Tabel 3.1 Daftar Aset

Gambar	Penjelasan	Sumber atau Pemilik
	Kinect V2 MS-SDK merupakan development kit untuk menghubungkan *game* dengan Kinect V2	RF Solutions (Beli)
	Boxing *Animation* Package digunakan sebagai animasi karakter 3D untuk *Battle* dan pemberian informasi grafik gerakan karakter 3D.	Waroath (Beli)
	Desain 3D Lampu ring *boxing* sebagai variasi untuk membangun environment yang menarik.	Aset pribadi
	Desain 3D Ring Tinju yang akan digunakan untuk tempat pertarungan karakter 3D dan pemberian informasi grafik gerakan karakter 3D.	Aset pribadi
	Desain 2D Sarung tinju yang digunakan sebagai indikator pemilihan menu dan spinning icon pada *Loading* scene.	Aset pribadi

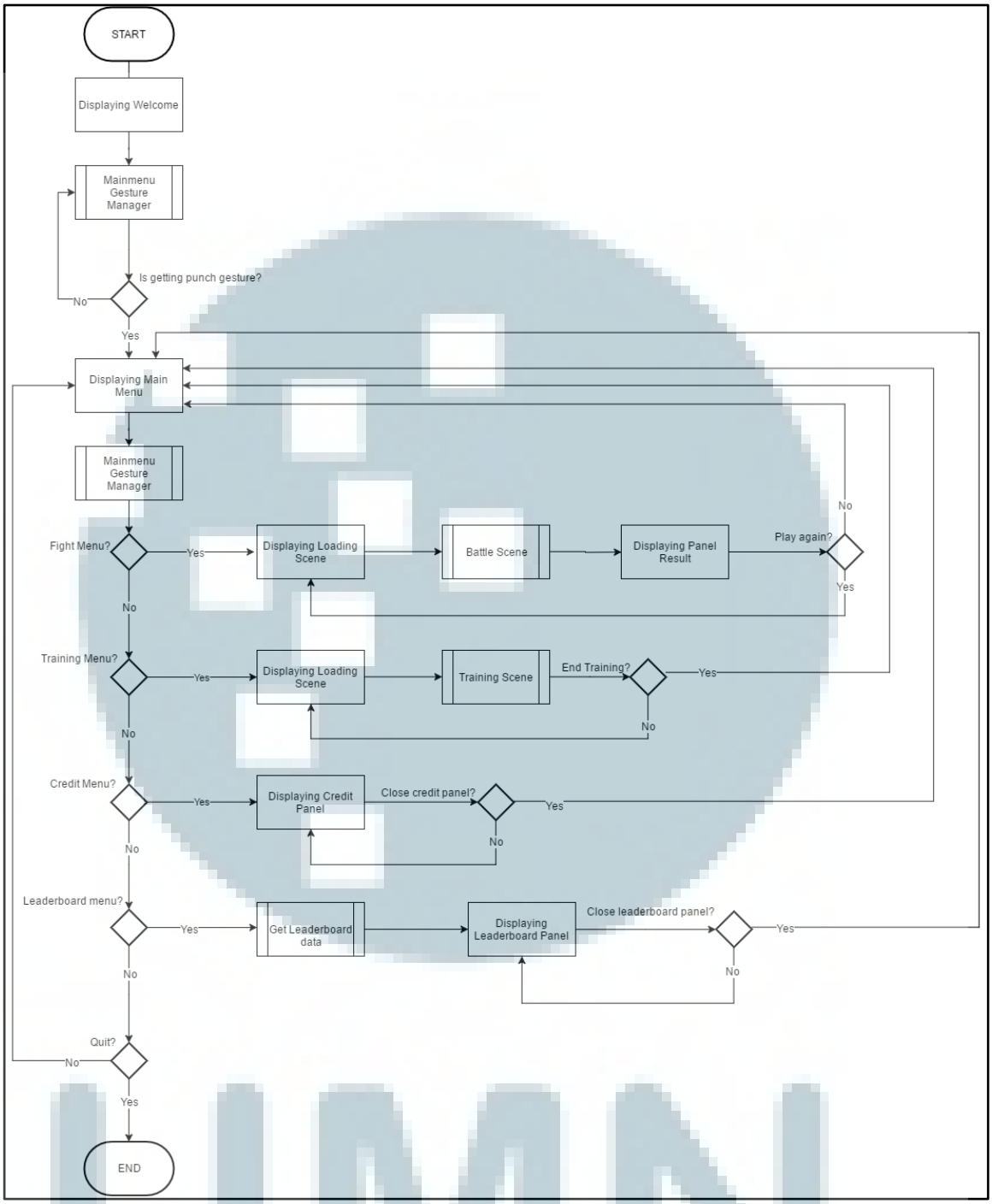
Tabel 3.1 Daftar Aset (Lanjutan)

Gambar	Penjelasan	Sumber atau Pemilik
	Desain 2D samsak yang digunakan untuk mempercantik panel-panel pada scene dan menjadi salah satu bagian dari icon *game* boxing action.	Aset pribadi
	Desain 3D kursi merah yang digunakan untuk memberikan variasi tambahan pada *game* environment.	Aset pribadi
	Desain 3D kursi biru yang digunakan untuk memberikan variasi tambahan pada *game* environment.	Aset pribadi
	Desain 3D meja yang digunakan untuk memberikan variasi pada *game* environment yang akan dijadikan tempat untuk meletakkan televisi.	Aset pribadi
	Desain 3D televisi yang digunakan untuk menampilkan menu-menu pada *game*.	Aset pribadi
	Desain 3D tiang yang digunakan untuk membangun tempat penyangga billboard pada *game* environment.	Aset pribadi

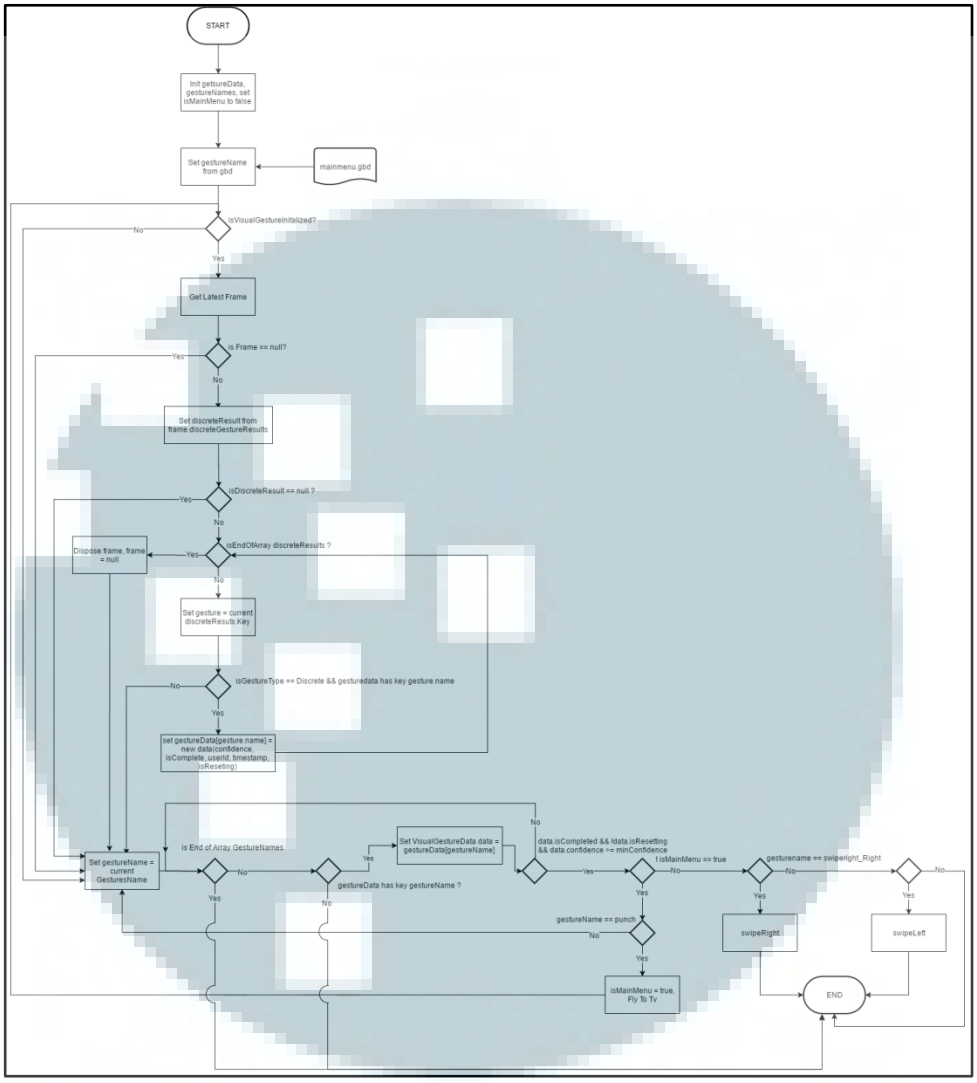
3.4 Perancangan Sistem

Perancangan *flowchart* dibuat untuk memperlihatkan alur sistem yang menjadi landasan pemograman secara keseluruhan. Gambar 3.1 merupakan *flowchart* umum *game*. Pada Gambar 3.1 saat *game* dimulai akan ditampilkan *welcome scene*. Untuk memasuki *mainmenu scene*, *player* diminta untuk melakukan gerakan *punch*. Setelah melakukan gerakan *punch*, sensor akan mendeteksi gerakan tersebut. jika gerakan benar, *game* akan secara otomatis memainkan animasi kamera untuk masuk ke dalam *mainmenu scene*. Pada *mainmenu scene player* akan diberikan 4 pilihan menu yang akan ditampilkan secara satu persatu mulai dari menu *battle* yang berada di posisi kiri kemudian menu *training* di posisi tengah dan menu *credit* di posisi kanan dan yang terakhir menu keluar yang terletak dipojok kiri layar. Ketika selesai milih menu, *player* dapat menggerakkan kursor dengan menggunakan tangannya dan untuk menekan seperti pada *mouse* dapat melakukan gerakan mengenggam.

Jika *player* memilih menu *battle*, *player* akan bertarung melawan komputer. Jika *player* memilih menu *training*, *player* akan diberikan informasi mengenai gerakan serangan dan pertahanan pada *boxing* dan akan ditampilkan bagaimana gerakan serangan atau pertahanan tersebut dan dapat mencoba gerakan tersebut juga. Jika *player* memilih menu c*redit*, *player* akan ditampilkan panel *credit* yang berisi individu yang terkait dalam proses perancangan dan pembangunan *game*. Jika *player* memilih menu *leaderboard*, *player* akan diberikan tampilan *leaderboard* pemegang *score* tertinggi. Jika *player* memilih menu keluar akan dikeluarkan dari *game*.



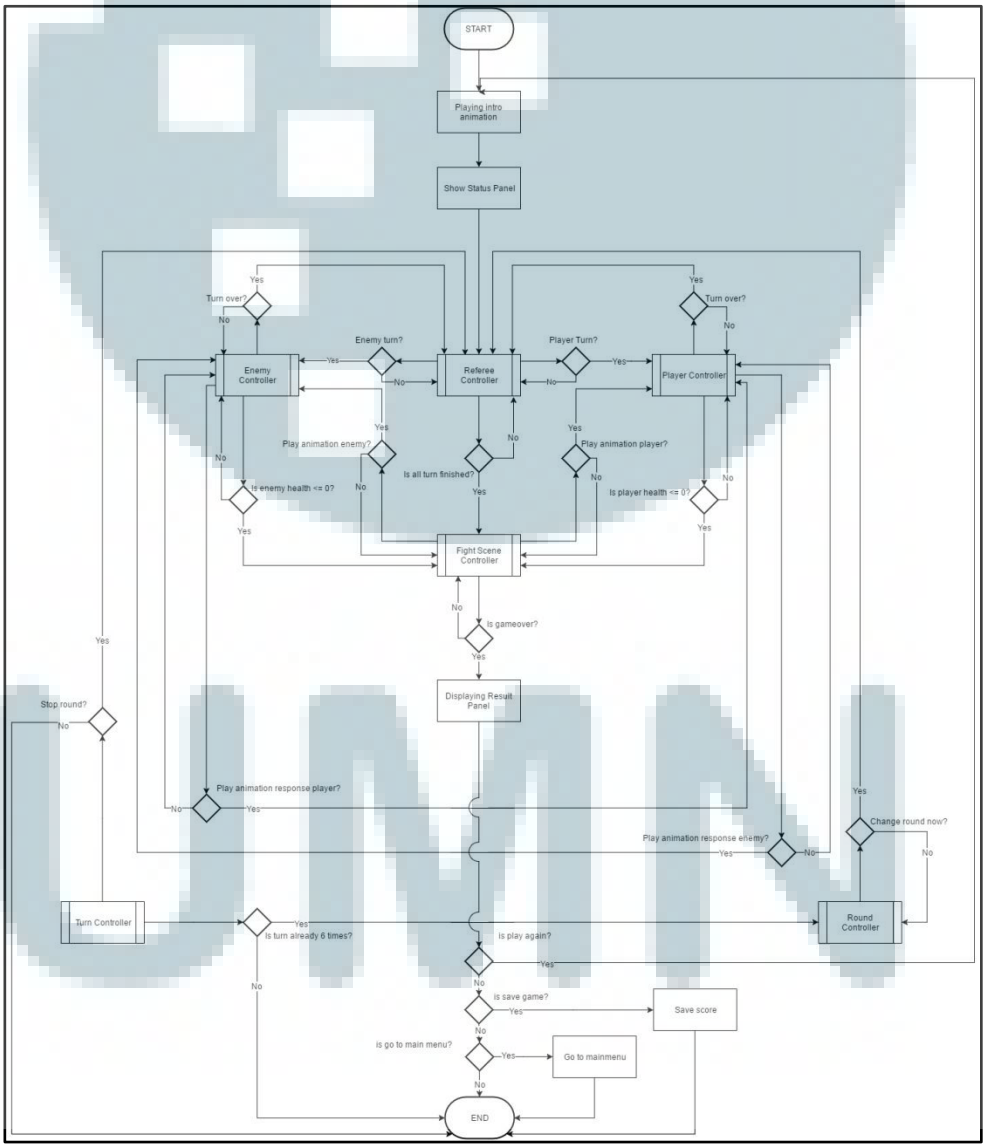
Gambar 3.1 *Flowchart* Umum *Game*



Gambar 3.2 *Flowchart Main Menu Gesture Manager*

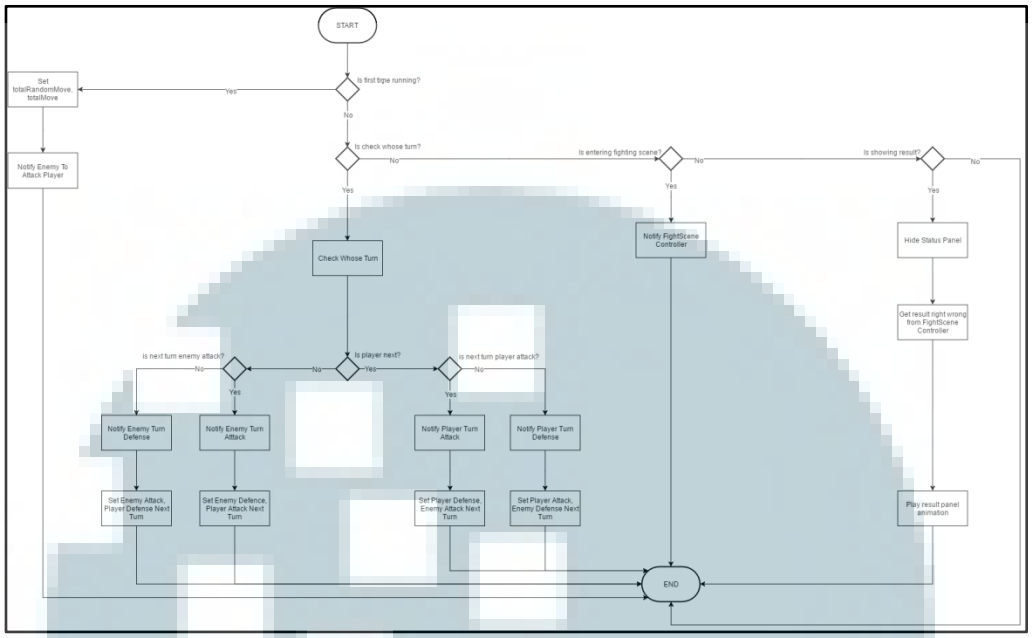
Gambar 3.2 menunjukkan alur sistem *main menu gesture manager*. Saat pertama kali proses berjalan. Proses inisialisasi *gestureData*, *gestureName*, dan *isMainMenu* akan dilakukan kemudian *mainmenu.gbd*, *file* yang menyimpan data *gesture*, akan di-*load* isinya. Setelah itu akan dilakukan pengecekan, apakah *visual gesture* sudah sudah diinisialisasi. Jika belum diinisialisasi, hentikan proses. Jika sudah diinisialisasi, proses akan mengambil *frame* rekaman terakhir yang direkam menggunakan sensor Kinect. Jika *frame* tidak kosong, akan diambil

discrete result-nya dan dilakukan pengecekan lagi apakah *result* kosong atau tidak. Jika *result* tidak kosong, akan dilakukan proses *loop* yang akan menentukan *gesture* yang diberikan *player*. Perulangan yang dilakukan sebanyak data yang terdapat di *discrete result* kemudian akan dilakukan pengecekan pada *gesture* yang diterima, apakah *gesture* tersebut sudah terdaftar dalam data yang di-*load* dari *mainmenu.gbd*. Jika *gesture* ditemukan di dalam *mainmenu.gbd*, isi *gestureData* dengan gerakan yang telah terdeteksi tersebut dengan isi *confidence*, *isComplete*, *playerId*, *timestamp*, dan *isReseting*.



Gambar 3.3 *Flowchart Battle Scene*

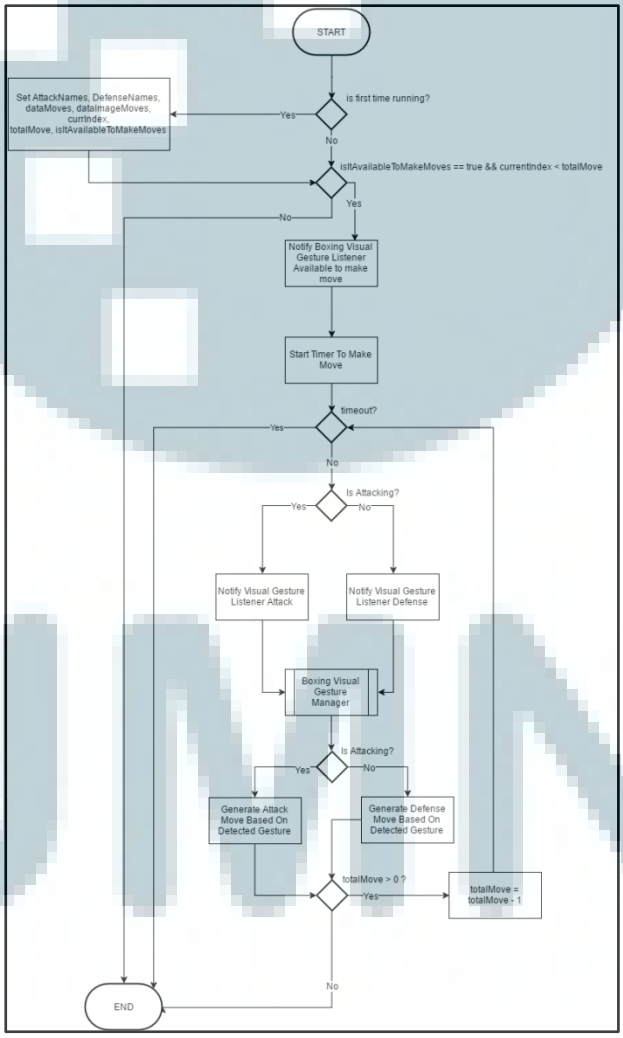
Gambar 3.3 merupakan *flowchart battle scene*, dapat dilihat pada awal mulai *battle scene* akan dimainkan animasi *intro* kamera yang akan mengelilingi arena *boxing* kemudian setelah animasi selesai, panel yang berisikan informasi mengenai *health point player* dan *health point enemy*, waktu, ronde, kalimat perintah akan ditampilkan. Pada awal *game referee controller* yang bertugas sebagai pemberitahu giliran siapa sekarang akan memberitahu *enemy* untuk jalan terlebih dahulu. Jika sekarang giliran *enemy*, *referee controller* akan memberitahu *enemy controller* bahwa sekarang gilirannya dan tipe gilirannya menyerang atau bertahan. Setelah giliran *enemy* selesai *enemy controller* akan memberitahu kepada *referee controller* bahwa gilirannya selesai dan *referee controller* akan memberitahu *player controller* bahwa sekarang adalah giliran *player*, tipe giliran *enemy* dan *player* selalu berbeda. Jika *enemy* menyerang, *player* bertahan dan jika *enemy* bertahan, *player* menyerang. Ketika giliran *player* sudah selesai maka *player controller* akan memberitahu ke *referee controller* bahwa gilirannya telah selesai. Jika masing-masing *enemy* dan *player* telah mendapatkan gilirannya *referee controller* akan memberitahukan ke *fight scene controller* untuk dimainkan animasi gerakan-gerakan yang dibuat oleh *enemy* dan *player*. Ketika e*nemy* melakukan serangan terdapat *function* di animasi yang akan memanggil *response* dari serangan tersebut, misalnya jika serangan *straight left jab*, *response* terkenanya adalah kepala terhentak kebelakang. Sebaliknya begitu juga jika *player* menyerang, *response* efek serangan di e*nemy* akan dipanggil dari *player*, proses ini akan terus berlanjut hingga salah satu dari *enemy* atau *player* kehabisan *health point*-nya terlebih dahulu atau ronde sudah mencapai 12. Ketika *game* selesai *player* dapat memilih bermain lagi atau simpan skor atau ke *mainmenu*.



Gambar 3.4 *Flowchart Referee Controller*

Gambar 3.4 menunjukkan alur sistem dari *referee controller*. Jika baru pertama kali berjalan set *variable totalrandommove* dan *totalmove* kemudian beritahu e*nemy* untuk menyerang *player*. Jika sudah pernah berjalan, cek apakah akan melakukan pengecekan giliran siapa kemudian jika giliran *player*, lakukan pengecekan lagi apakah giliran selanjutnya adalah serangan, jika giliran selanjutnya adalah serangan, beritahu *player* untuk menyerang kemudian set *player defense* dan *enemy attack* untuk giliran kedepannya lagi. Jika tidak beritahu *player* untuk melakukan gerakan *defense* dan beritahu juga giliran selanjutnya *player* akan menyerang dan *enemy* akan bertahanan. Jika giliran *enemy*, sama seperti *player* dilakukan pengecekan lagi apakah giliran selanjutnya adalah serangan, jika giliran selanjutnya serangan, beritahu e*nemy* untuk menyerang dan beritahu juga giliran selanjutnya *enemy* akan bertahanan dan *player* menyerang, jika giliran selanjutnya ada bertahan, beritahu e*nemy* untuk bertahan kemudian

beritahu juga giliran selanjutnya *enemy* akan menyerang dan *player* akan bertahan. Jika tidak melakukan pengecekan giliran siapa, cek apakah saatnya untuk memberitahu *fight scene controller* untuk memulai animasi pertarungan. Jika bukan saatnya untuk memberitahu *fight scene controller* untuk memulai animasi, beritahu *fight scene controller* untuk cek apakah saatnya untuk menampilkan hasil pertarungan. Jika sekarang saatnya untuk menampilkan hasil pertarungan, sembunyikan panel status dan ambil hasil pertarungan dan tampilkan panel *result*. Jika akan memulai animasi, beritahu *fight scene controller* untuk memainkan animasi pertarungan.



Gambar 3.5 *Flowchart Player Controller*

Gambar 3.5 menunjukkan alur sistem dari *player controller*. Pada awal pertama *player controller* berjalan, akan dilakukan insialisasi beberapa komponen seperti *array attacknames* dan *defensenames* dan beberapa yang lainya. Jika sudah pernah berjalan, akan dilakukan pengecekan apakah sekarang saatnya untuk memasukkan *input* dan apakah jumlah *input* masih kurang dari *totalmove* yang diperbolehkan. Jika jumlah input sudah lebih dari *totalmove*, proses berhenti. Jika jumlah *input* masih kurang dari *totalmove*, beritahu *boxing gesture manager* untuk memperbolehkan *player* melakukan *gesture* serangan atau pertahanan sesuai dengan giliran yang akan diberitahukan oleh *player controller*.

Setelah itu, jalankan *timer* untuk membatasi lama penginputan *gesture* dari *player*. Setelah itu, lakukan pengecekan apakah *timer* sudah sampai batas. Jika sudah sampai batas, berhenti mengambil *input*-an gerakan dari *player*. Jika *timer* belum melewati batas, lakukan pengecekan tipe giliran *player* sekarang menyerang atau bertahan.

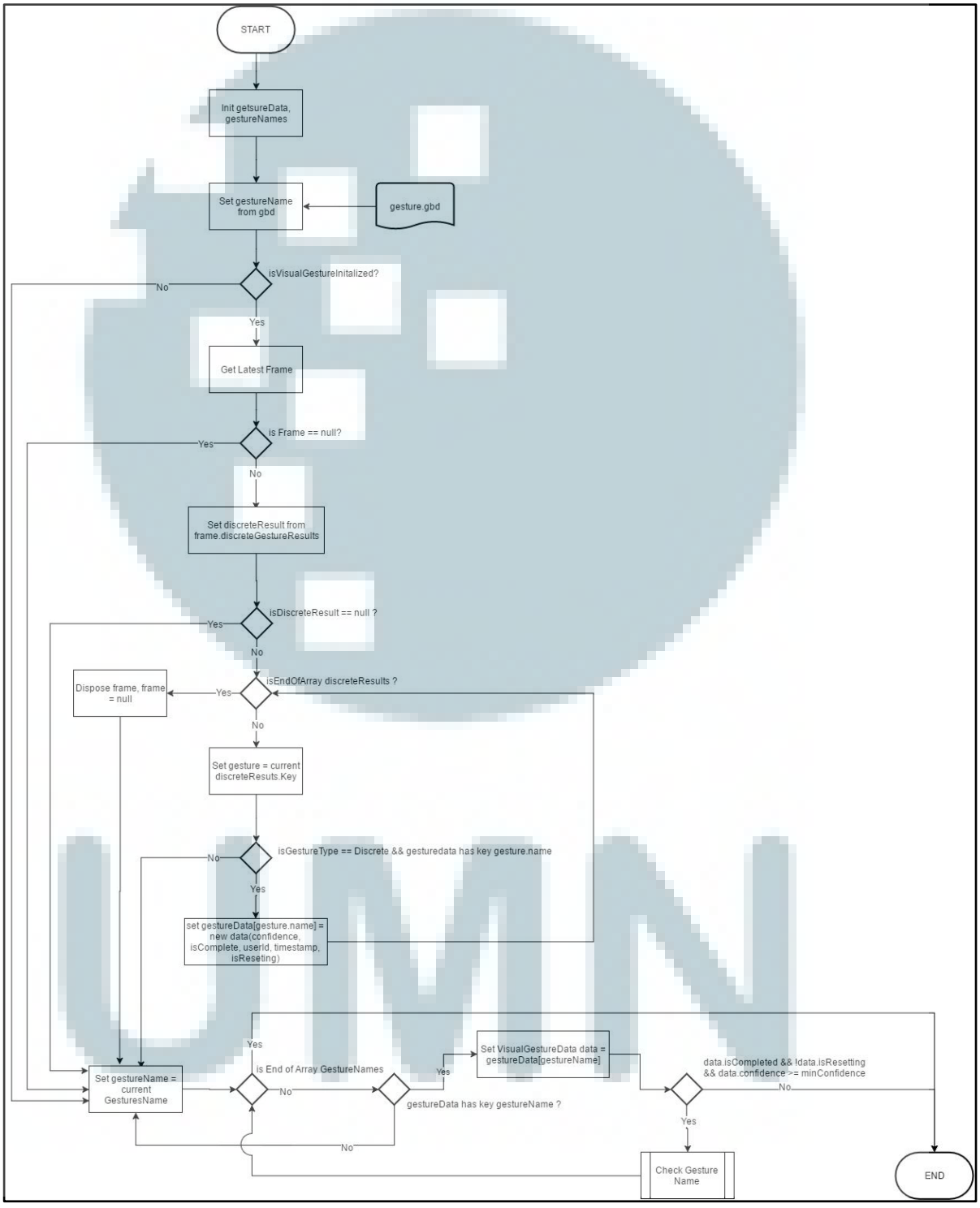
Jika tipe giliran *player* menyerang, beritahu *boxing gesture manager* untuk mengambil gerakan menyerang saja, begitu juga sebaliknya jika tipe giliran *player* bertahan, beritahu *boxing gesture manager* untuk mengambil gerakan pertahanan saja. Saat proses deteksi berhasil *boxing gesture manager* akan memberitahukan ke *player controller* hasil *input* dari *player* yang diambil dengan menggunakan sensor Kinect apakah hasil *input* yang diberikan *player* itu serangan atau pertahanan. Jika hasil *input player* serangan, berikan gambar serangan yang tadi di-*input* oleh *player*. Jika hasil *input player* bertahan, berikan gambar gerakan bertahan yang tadi dimasukkan oleh *player*.

Setelah itu, dilakukan pengecekan lagi apakah jumlah *totalmove* kurang dari 1. Jika *totalmove* belum kurang dari 1, ulangi proses yang dijabarkan di atas sampai *totalmove* kurang dari 1. Gambar 3.6 menunjukkan alur sistem dari *boxing gesture manager*. Proses awal yang berjalan adalah inisialisasi *gestureData*, *gestureName*, dan *isGrantedToMove* akan dilakukan kemudian *gesture.gbd*, *file* yang menyimpan data *gesture*, akan di-*load* isinya.

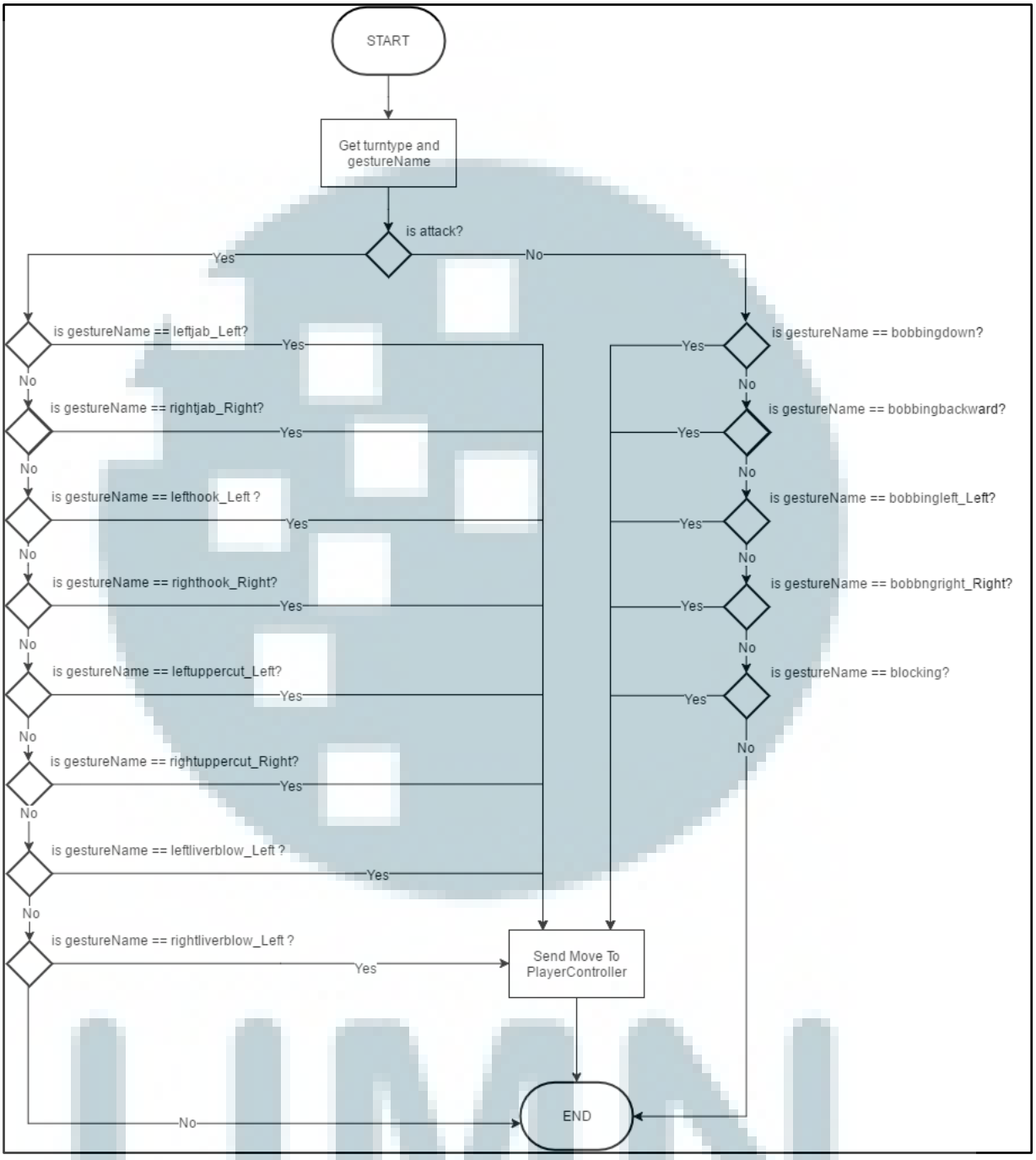
Setelah itu akan dilakukan pengecekan apakah *visual gesture* sudah sudah diinisialisasi. Jika belum diinisialisasi, hentikan proses. Jika sudah diinisialisasi, proses akan mengambil *frame* rekaman terakhir yang direkam menggunakan sensor Kinect. Jika *frame* tidak kosong, akan diambil *discrete result*-nya dan dilakukan pengecekan lagi apakah *result* kosong atau tidak. Jika *result* tidak kosong, akan dilakukan proses *looping* yang akan menentukan *gesture* yang diberikan *player*. Perulangan yang dilakukan sebanyak data yang terdapat di *discrete result* kemudian akan dilakukan pengecekan pada *gesture* yang diterima, apakah *gesture* tersebut sudah terdaftar dalam *gesture.gbd*.

Jika *gesture* tidak terdaftar dalam *gesture.gbd*, hentikan proses pemeriksaan. Jika *gesture* terdaftar dalam *gesture.gbd*, proses *check gesture name* akan memeriksa gerakan apakah yang di-*input* oleh player. Proses ini akan terus berlangsung hingga *array discrete result* sudah sampai di *index* paling akhir. Proses *check gesture name* dapat dilihat pada Gambar 3.7, proses akan memeriksa tipe giliran terlebih dahulu. Jika tipe giliran menyerang, cek gerakan serangan saja. Jika tipe giliran bertahan, cek gerakan bertahan saja.

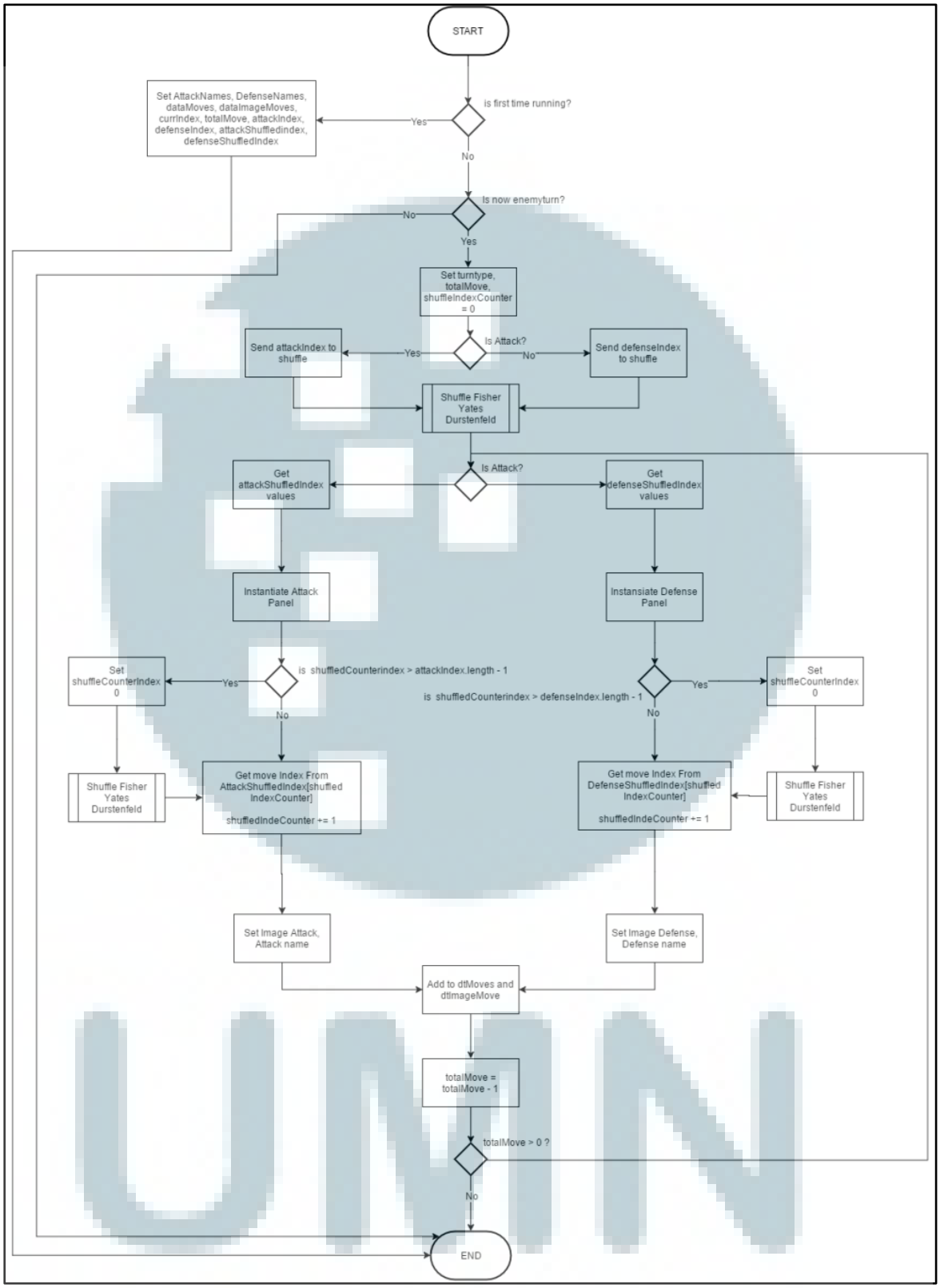
Jika gerakan yang dilakukan oleh player tidak terdaftar, hentikan proses pengecekan. Jika gerakan yang dilakukan oleh player terdaftar, kirim gerakan tersebut ke *player controller* dan setelah itu hentikan proses pengecekan.



Gambar 3.6 *Flowchart Boxing Gesture Manager*



Gambar 3.7 *Flowchart Check Gesture Name*



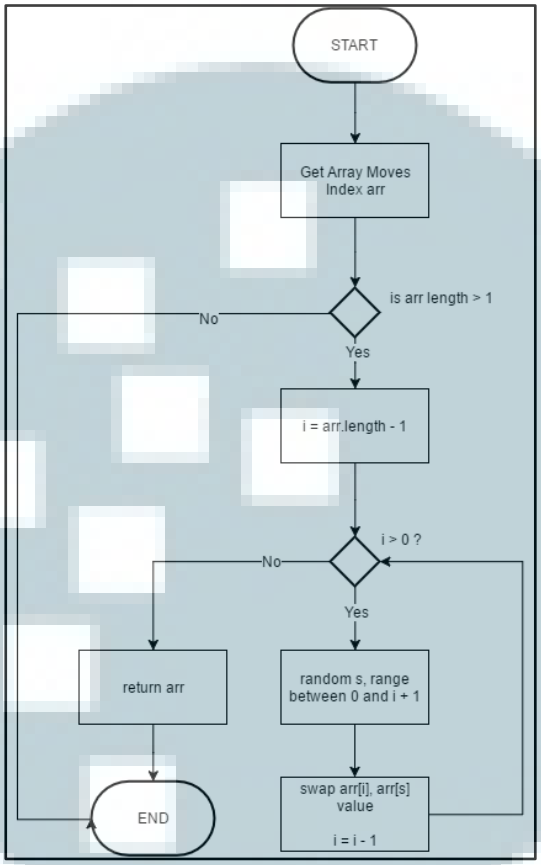
Gambar 3.8 *Flowchart Enemy Controller*

Gambar 3.8 menunjukkan alur sistem dari *enemy controller*. Pada awal pertama *enemy controller* berjalan, akan dilakukan insialisasi beberapa komponen seperti *array attacknames* dan *defensenames* dan lain-lainnya. Jika sudah pernah berjalan, cek apakah sekarang giliran *enemy*. Jika sekarang bukan giliran *enemy*, hentikan proses. Jika sekarang giliran *enemy*, set *turntype* dan *totalmove*. *Turntype* merupakan *variable* yang menyimpan tipe giliran e*nemy* menyerang atau bertahan dan set *shuffledIndexCounter*.

Setelah itu dilakukan pengecekan apakah *enemy* menyerang. Jika *enemy* menyerang, kirim *attackIndex* ke *function FisherYatesDurstenfeld* untuk dilakukan pengacakan posisi dan jika *enemy* bertahan, kirim *defenseIndex* ke *function FisherYatesDurstenfeld* untuk dilakukan pengacakan juga. Setelah itu, dilakukan pengecekan tipe giliran lagi, jika *enemy* menyerang, ambil nilai dari hasil pengacakan *attackShuffledIndex* dan jika *enemy* bertahan, ambil nilai dari hasil pengacakan *defenseShuffledIndex*.

Jika *enemy* menyerang, buat *panel* menyerang. Jika *enemy* bertahan, buat *panel* bertahan. Setelah membuat *panel*, akan dilakukan pengecekan jumlah *shuffleIndexCounter*. Jika *shuffleIndexCounter* nilainya lebih besar dari panjang *attackIndex* dikurang 1, lakukan *shuffling* lagi, pengecekan *shuffleIndexCounter* dibandingkan dengan *array attackIndex* kalau tipe giliran menyerang dan *array defenseIndex* kalau tipe giliran bertahan. Jika *shuffleIndexCounter* tidak lebih besar dari *attackIndex* dikurang 1, ambil nilai dari *attackShuffledIndex* dengan index *shuffleIndexCounter* kemudian tambah *shuffleIndexCounter* dengan 1. Setelah itu, lakukan pengecekan apakah *totalmove* sudah sama dengan 0.

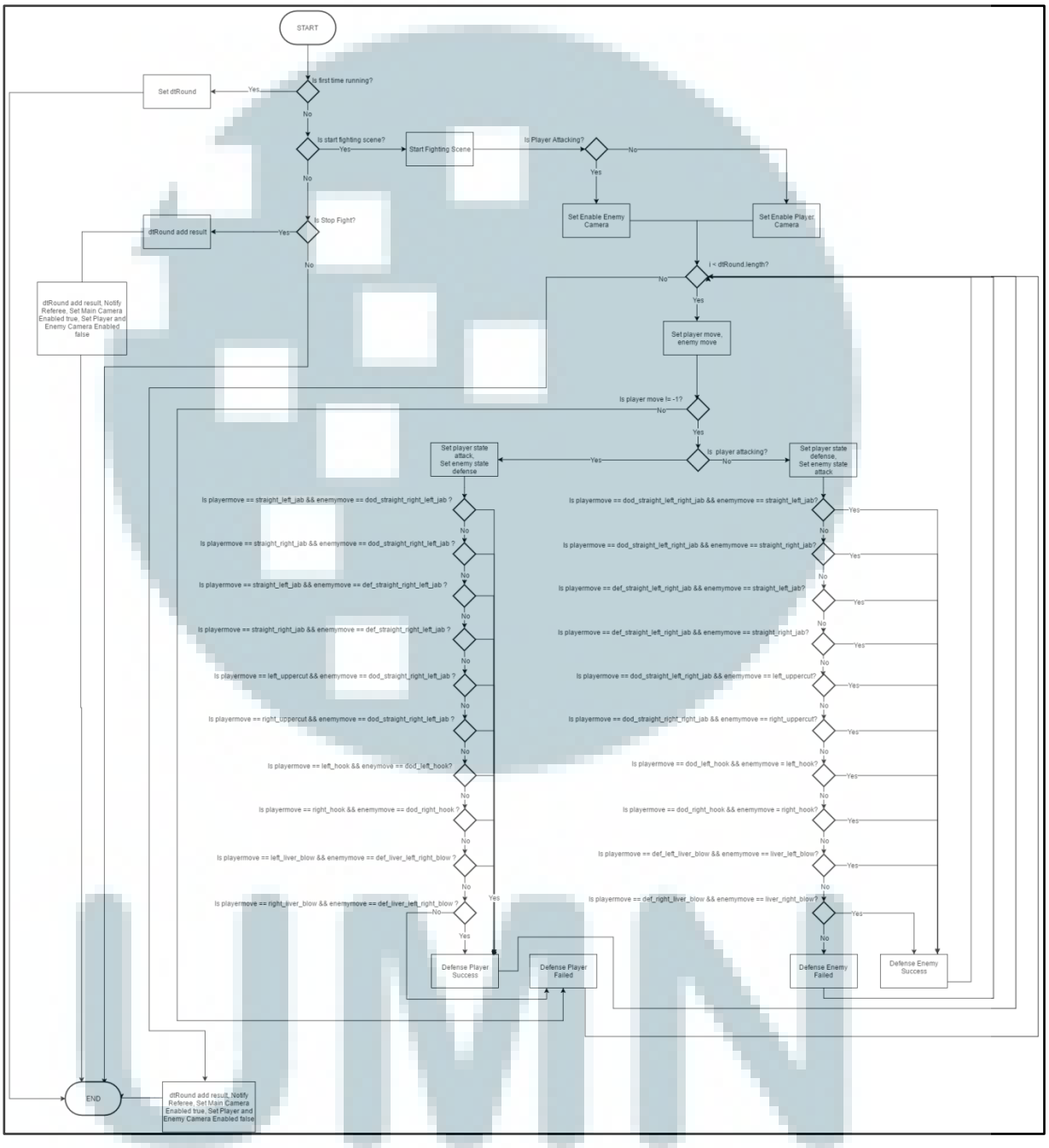
Jika *totalmove* sudah sama dengan 0, hentikan proses. Jika *totalmove* belum sama dengan 0, ulangi proses di atas hingga *totalmove* sama dengan 0.



Gambar 3.9 *Flowchart Shuffle* Fisher-Yates-Durstenfeld.

Gambar 3.9 menunjukkan *flowchart* algoritma *shuffle* Fisher-Yates-Durstenfeld, proses awal yang dilakukan adalah mengambil nilai *array* yang berisikan *move index* kemudian dilakukan pengecekan apakah panjang *array* tersebut lebih dari 1. Jika panjang *array* lebih dari 1, set *variable i* dengan panjang *array* dikurang 1. Setelah itu, lakukan perulangan untuk mengacak *array*. Proses awal pengacakan array adalah melakukan *random variable s* yang akan digunakan sebagai *index array* kemudian menukar nilai *array index variable i* dengan *index variable s*. Proses ini akan dilakukan sampai *variable i* kurang dari 0. Setelah itu,

function akan memberikan *array* yang telah diacak tersebut ke bagian *game* yang telah memanggilnya.



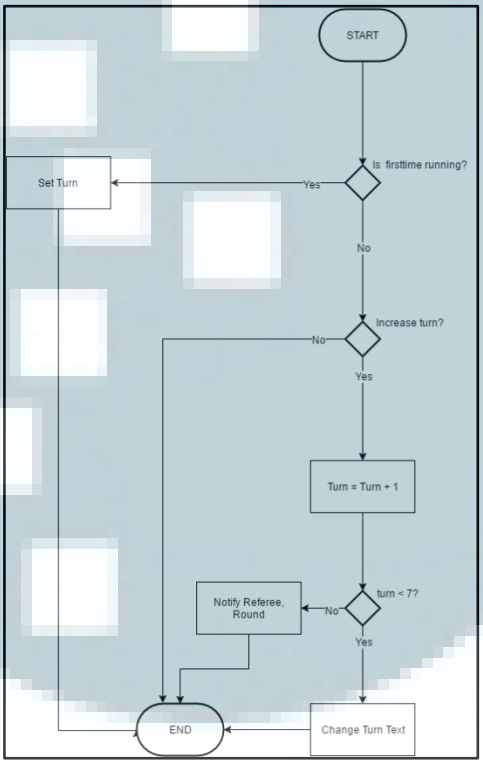
Gambar 3.10 *Flowchart Fight Scene Controller*

Gambar 3.10 menunjukan alur sistem dari *fight scene controller*. Pada awal jika proses baru berjalan pertama kali, dilakukan inisialisasi *dtRound* yang akan menampung hasil tiap-tiap ronde. Jika sudah pernah berjalan akan dicek apakah akan memulai *fight scene*. Jika tidak memulai *fight scene* akan dilakukan pegecekan selanjutnya, yaitu pengecekan apakah akan menyelesaikan ronde.

Jika ronde akan selesai, data hasil pertarungan akan dimasukkan ke dalam *dtRound* dan beritahu *referee controller* kalau ronde telah selesai. Jika memasuki *fight scene*, lakukan pengecekan apakah *player* mendapatkan tipe giliran serangan. Jika *player* mendapatkan tipe giliran serangan *player* menyerang, aktifkan kamera e*nemy* agar dari sisi *player* dapat melihat gerakan serangan tersebut. Jika *player* mendapatkan tipe giliran bertahan, aktifkan kamera *player*. Setelah itu dilakukan *loop* pada proses dengan mengiterasi *variable i* yang akan selalu ditambah 1 hingga lebih dari jumlah data *dtRound*.

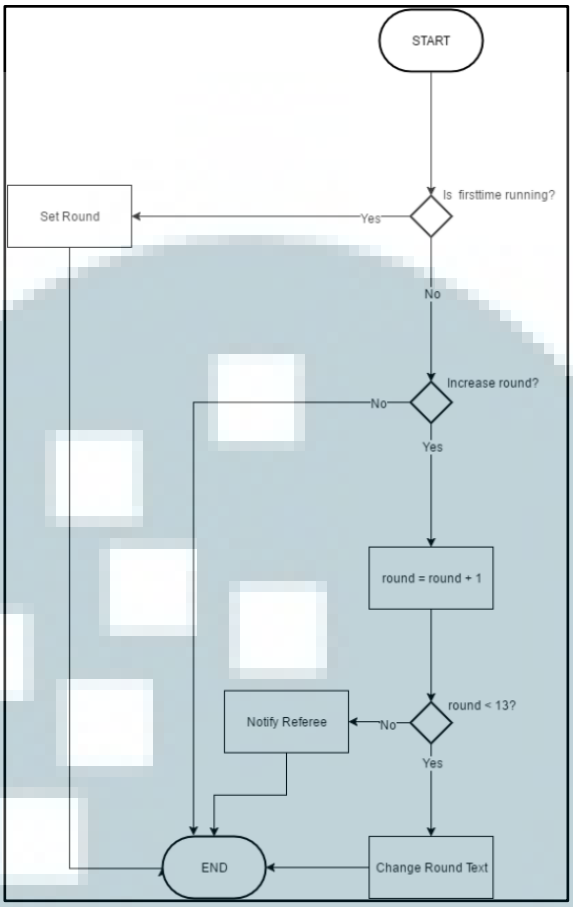
Setelah itu, akan diset *playermove* dan *enemymove* sesuai dengan data yang ada di *dtRound*. Setelah itu akan dilakukan pengecekan terlebih dahulu apakah *playermove* sama dengan -1 yang berati *player* tidak memasukkan input pada saat pertarungan. Jika nilai *playermove* sama dengan -1, langsung mainkan animasi *player* tertinju yang sesuai dengan gerakan tinjunya. Jika *playermove* tidak sama dengan -1, akan dilakukan pengecekan apakah pada saat itu *player* sedang menyerang atau bertahan. Jika *player* bertahan lakukan pengecekan apakan *move* bertahan *player* sudah cocok untuk menangkis *move* serangan dari e*nemy*. Jika tidak cocok *move*-nya berati gagal melakukan pertahanan. Begitu juga sebaliknya jika *player* menyerang lakukan pegecekan *move* menyerang *player* apakah bisa ditangkis dengan *move* bertahan *enemy*.

Jika tidak bisa ditangkis oleh *enemy*, berati gerakan serangan dari *player* benar. Ulangi semua proses di atas sampai jumlah iterasi *variable i* sama dengan atau lebih dari banyak data *dtRound*. Setelah selesai, simpan hasil pertarungan dan beritahu *referee controller* animasi pertarungan telah selesai dan set kamera utama lagi menjadi aktif dan set kamera *player* dan kamera e*nemy* menjadi tidak aktif.



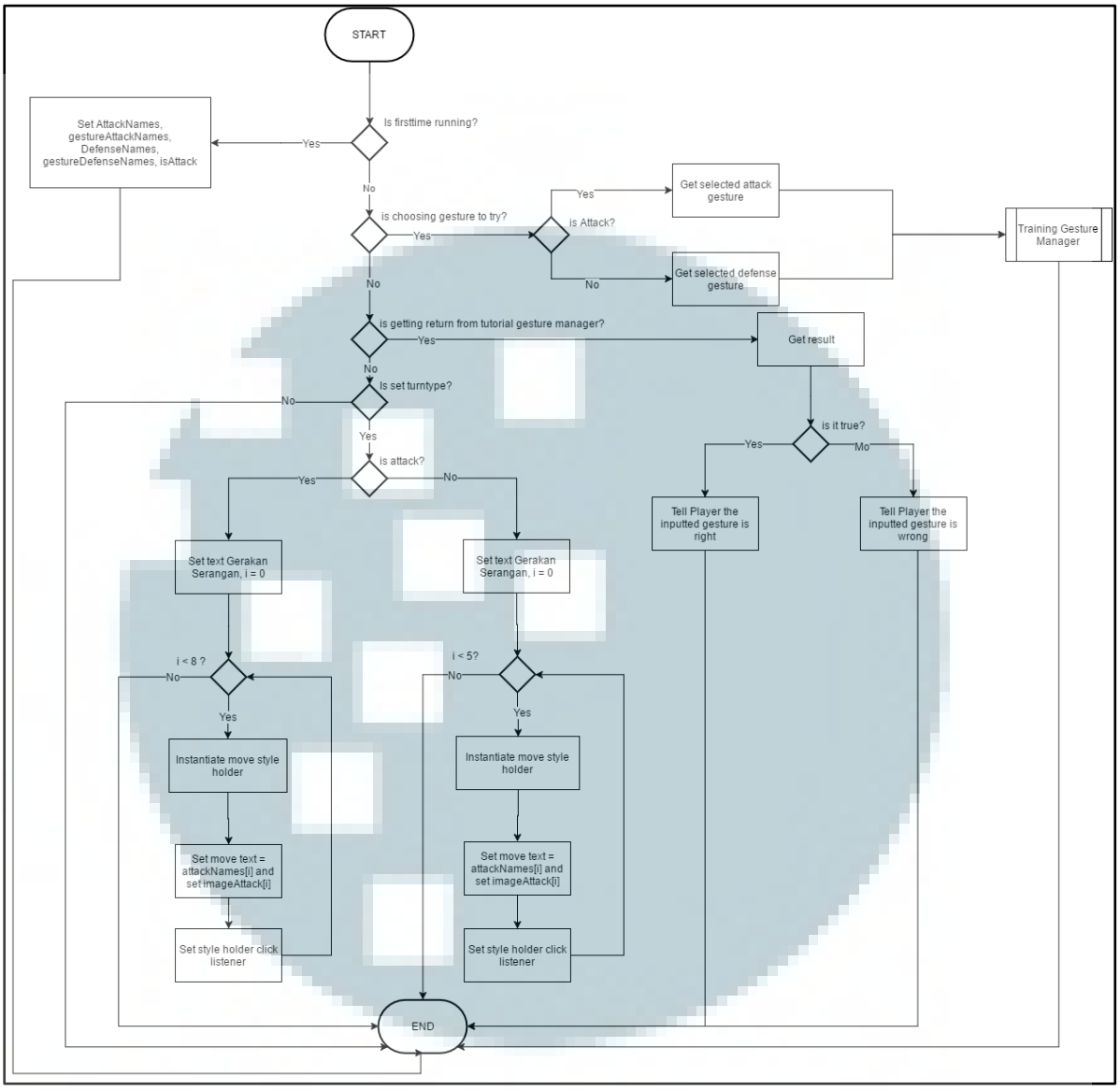
Gambar 3.11 *Flowchart Turn Controller*

Gambar 3.11 menunjukkan alur sistem *turn controller*. Pertama akan dilakukan pengecekan apakah baru berjalan pertama kali. Jika baru pertama kali berjalan, lakukan insialisasi set *turn*. Jika sudah pernah berjalan, lakukan pengecekan apakah tambah *turn*. Jika tambah *turn*, tambah *turn* dan lakukan pengecekan lagi apakah *turn* sudah 7. Jika belum kurang dari 7, ubah teks *turn* untuk menampilkan *turn* keberapa sekarang. Jika *turn* sudah 7, beritahu *referee controller* dan *round controller*.

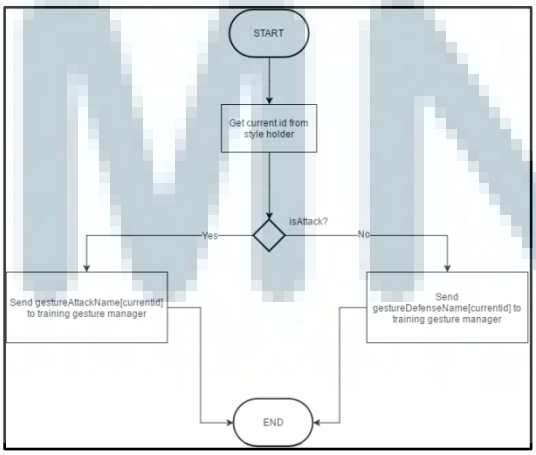


Gambar 3.12 *Flowchart Round Controller*

Gambar 3.12 menunjukkan alur sistem dari *round controller*. Saat pertama kali berjalan, akan dilakukan inisialisi set *round*. Jika sudah pernah berjalan, dilakukan pengecekan apakah sekarang saatnya untuk menambah *round*, penambahan *round* dilakukan jika ada pemberitahuan dari *turn controller*. Jika ada pemberitahuan dari *turn controller*, tambah *round* kemudian lakukan pengecekan apakah *round* sudah 13. Jika *round* belum sampai ke 13, ubah teks *round* untuk memberitahu *player* perubahan ronde di layar *game*. Jika ronde sudah 13, beritahu ke *referee controller* untuk menghentikan *game*.



Gambar 3.13 *Flowchart Training Controller*



Gambar 3.14 *Flowchart Style Holder Click Listener*

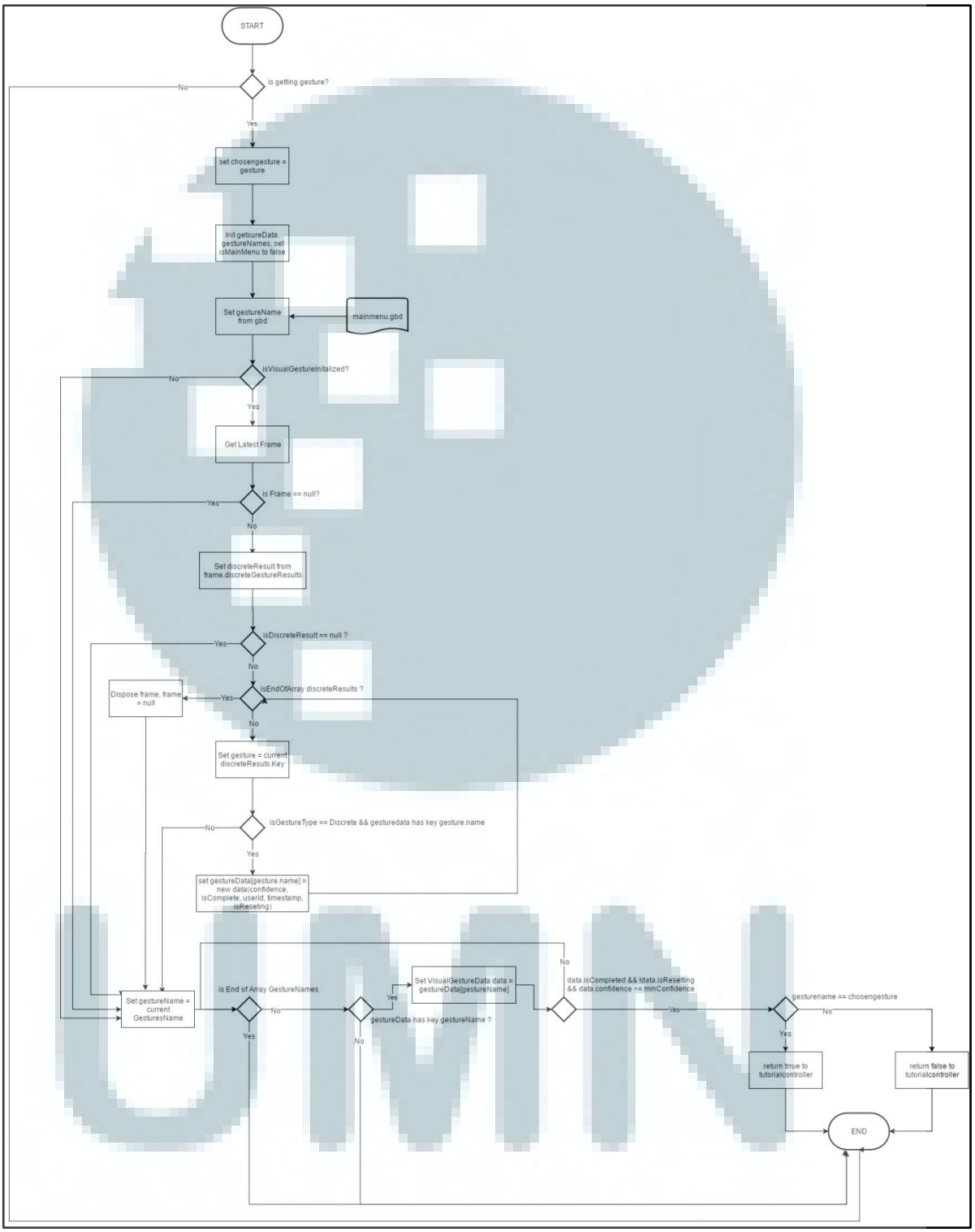
Gambar 3.13 menunjukkan alur sistem dari *training controller*. Jika baru pertama kali berjalan, lakukan inisialisasi *attacknames*, *gestureattacknames*, *defensenames*, *gesturedefensenames*, dan *isAttack*. Jika sudah pernah berjalan, dilakukan pengecekan apakah menerima *result* dari *training gesture manager*.

Jika menerima *result* dari *training gesture manager*, ambil *result*-nya lalu dicek apakah *result true* atau *false* dan beritahu ke *player* hasilnya kemudian hentikan proses. Jika tidak menerima *result* dari *training gesture manager*, lakukan pengecekan lagi apakah sedang menetapkan nilai *turntype*. Jika tidak sedang menetapkan nilai *turntype*, hentikan proses. Jika sedang menetapkan nilai *turntype*, lakukan pengecekan apakah *turntype* menyerang atau bertahan. Jika *turntype* menyerang, set isi text pada layar menjadi gerakan serangan kemudian set *variable i* jadi 0 lalu lakukan perulangan yang akan membuat objek yang menampung gerakan-gerakan dengan mengambil data dari *variable attacknames*.

Proses yang sama akan dilakukan ketika *turntype* bertahan, yang membedakan adalah data yang diambil jika *turntype* bertahan adalah dari *variable defensenames*. Setelah mengambil data selesai, implementasikan *click listener* kepada objek. Jika *turntype* menyerang, perulangan akan berjalan sebanyak 8 kali dan jika *turntype* bertahan, perulangan akan berjalan sebanyak 5 kali.

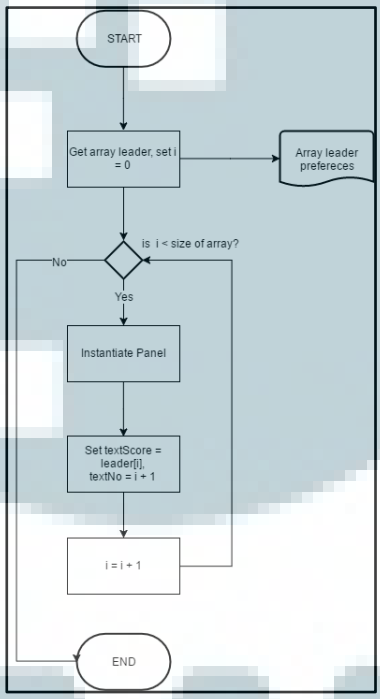
Gambar 3.14 menunjukkan alur sistem *style holder click listener* ketika tombol *style holder* ditekan, *listener* ini akan dipasangkan ke daftar gerakan yang ditampilkan di *training scene*. Saat *style holder* ditekan secara otomatis *click listener* akan mengambil *id* dari *button* yang ditekan tersebut lalu dilakukan pengecekan apakah gerakan tersebut serangan atau pertahanan Jika gerakan yang

dipilih serangan, kirim gerakan serangan tersebut yang diambil dari *array gestureAttackNames* dengan *index* id ke *training gesture manager*.



Gambar 3.15 *Flowchart Training Gesture Manager*

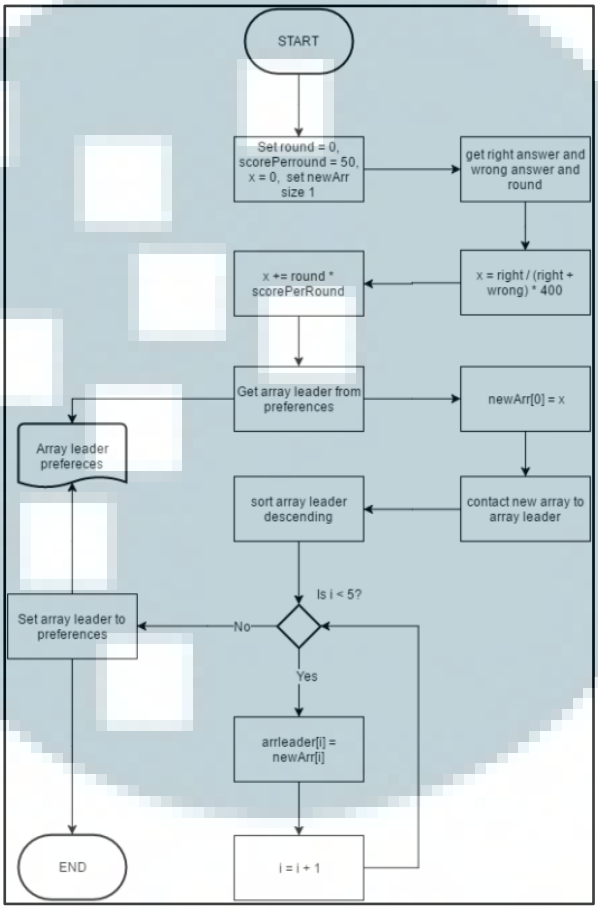
Gambar 3.15 menunjukkan alur sistem *training gesture manager*. Alur sistem hampir sama dengan *mainmenu gesture manager* dan *boxing gesture manager*. Proses yang berbeda hanya *training gesture manager* akan menerima *input* dari *player* yang akan dikirim dari *style holder click listener* lalu nilai input dari *player* itu akan dicek apakah gerakan dari yang dipilih *player* sama dengan gerakan yang *player* lakukan. Jika gerakan benar, beritahu *training controller* kalau gerakan benar dan jika gerakan salah, beritahu *training controller* kalau gerakan salah.



Gambar 3.16 *Flowchart Get Leaderboard Data*

Gambar 3.16 menunjukkan alur sistem dari *get leaderboard data*. Saat pertama kali berjalan akan dilakukan inisialisasi *nilai variable i* menjadi 0 dan akan diambil *array leader* dari *preferences*. Setelah itu, akan dilakukan perulangan untuk mencetak nilai 5 terbaik dari hasil pertarungan. Proses akan menanyakan apakah nilai *variable i* kurang dari panjang *array*. Jika nilai *variable*

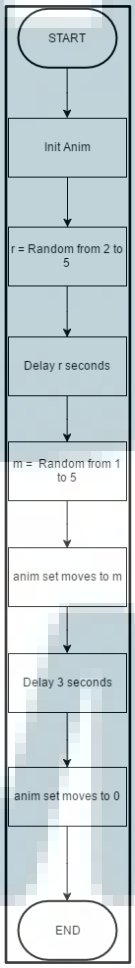
i kurang dari panjang *array*, buat sebuah panel dan isi teks yang berada di dalam panel dengan isi *array index* ke *variable i*. Setelah itu, tambahkan *variable i* dengan satu. Proses ini akan terus berjalan sampai nilai *variable i* lebih dari panjang *array*.



Gambar 3.17 *Flowchart* Penyimpanan *Battle Result*

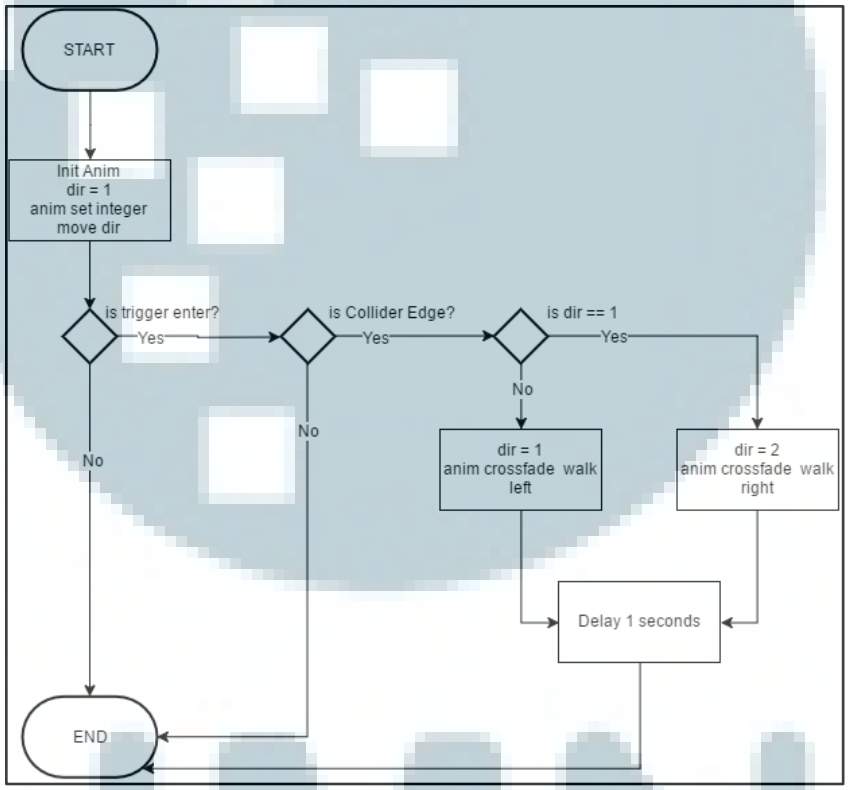
Gambar 3.17 menunjukkan alur sistem untuk menyimpan hasil pertarungan. Proses awal yang akan berjalan adalah inisialisasi *variable round* dan *x* menjadi 0 dan *scorePerRound* menjadi 50. Setelah itu, proses pengambilan *right answer*, *wrong answer*, dan *round* dilakukan kemudian perhitungan untuk mendapatkan *score player* dengan menghitung persentase *right answer* lalu menghitung *score* jumlah *round* yang telah diselesaikan *player*. Setelah proses

perhitungan selesai, maka proses akan mengambil data *array leader* dari *preferences* untuk mengambil nilai 5 terbaik dari hasil pertarungan. Setelah proses pengambilan *array leader* selesai, nilai perhitungan *score player* yang telah dilakukan sebelumnya akan dimasukkan ke dalam *array leader* kemudian dilakukan *sorting*. Setelah *sorting* selesai, akan dilakukan perulangan untuk mengambil nilai 5 terbaik lalu ketika perulangan telah selesai *array leader* akan dimasukkan ke dalam *preferences* lagi dan proses pengambilan *array leader* selesai.



Gambar 3.18 *Flowchart Taunt Spectator Controller*

Gambar 3.18 menunjukkan alur sistem dari *taunt spectator controller*. Proses awal yang berjalan adalah proses inisialisasi komponen *animator*, *animator* digunakan untuk mengatur perpindahan animasi. Setelah itu, mengambil angka acak yang akan digunakan sebagai *delay*. *Delay* digunakan agar animasi antar *spectator* tidak berjalan secara bersamaan kemudian mengambil angka acak lagi yang digunakan untuk memilih animasi yang akan dimainkan agar gerakan *spectator* bervariasi dari satu dengan lainnya.



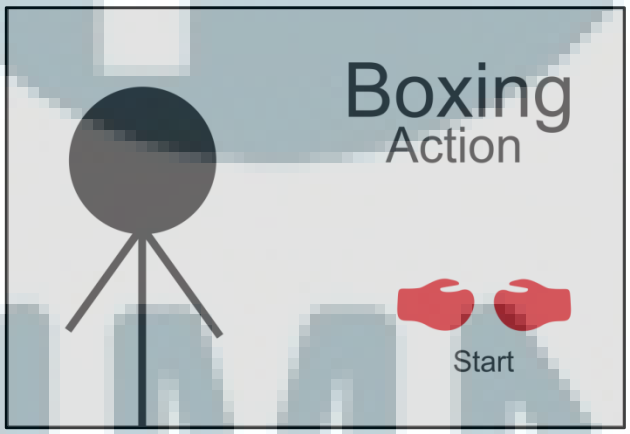
Gambar 3.19 *Flowchart Walking Spectator Controller*

Gambar 3.19 menunjukan alur sistem dari *walking spectator controller*. Pada awal proses inisialisasi komponen *animator*, *variable dir* menjadi 1 yang artinya berjalan ke arah kiri, dan set *variable move* dalam *animator* menjadi 1. Setelah proses inisialisasi selesai, lakukan pengecekan apakah memasuki area *trigger*. Jika tidak memasuki area *trigger*, hentikan proses. Jika memasuki area

trigger, lakukan pengecekan apakah *collider* dari *trigger* adalah *edge*. Jika bukan *edge*, hentikan proses. Jika *collider* dari *trigger* adalah *edge*, lakukan pengecekan apakah *variable dir* sama dengan 1. Jika *variable dir* sama dengan 1, ubah *variable dir* menjadi 2 dan ubah animasi menjadi berjalan ke arah kanan. Jika *variable dir* tidak sama dengan 1, ubah *variable dir* menjadi 2 dan ubah animasi menjadi berjalan ke kiri. Setelah pengecekan *dir* selesai, tunggu selama 1 detik dan hentikan proses.

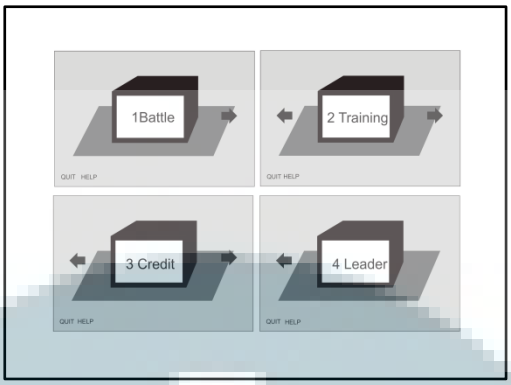
3.5 Perancangan Player Interface

Pembuatan rancangan *player interface* dibuat untuk menjabarkan fungsionalitas di tiap *player interface* dan gambaran mengenai tata letak kompenen di *player interface*. Adapun rancangan *player interface* untuk *welcome scene* seperti pada Gambar 3.20.



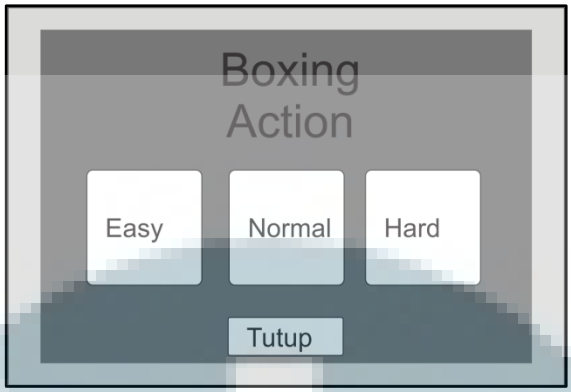
Gambar 3.20 *Mockup Player Interface Welcome Scene*

Gambar 3.20 menunjukkan *mockup* untuk tampilan antarmuka *mainmenu* pada *game*. Menu ini merupakan menu *introduction* pada *game* dan terdapat animasi *punching gloves* yang memberikan arahan ke *player* untuk melakukan *gesture punching* untuk masuk ke dalam *mainmenu scene*.

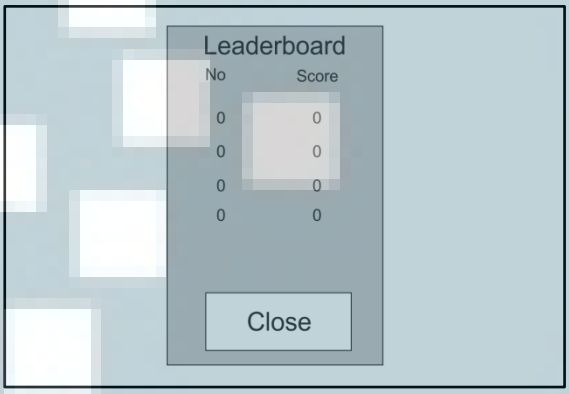


Gambar 3.21 *Mockup Player Interface Mainmenu Scene*

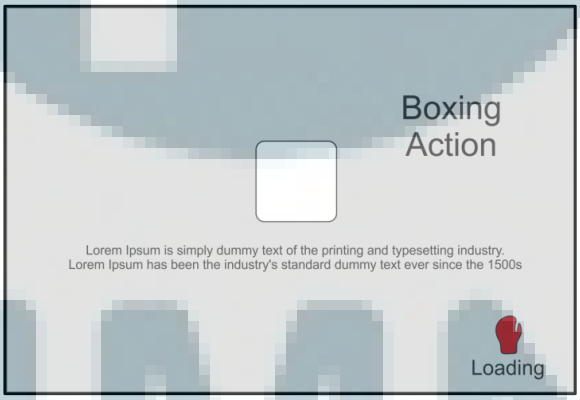
Gambar 3.21 menunjukkan *mockup* untuk tampilan antarmuka *mainmenu scene*. Terdapat 4 buah menu yang ditampilkan satu persatu dan dapat diganti dengan menggunakan *gesture swipe right* untuk menuju menu yang ada di sebelah kanan dan *swipe left* untuk menuju menu yang berada disebelah kiri. Menu pertama adalah menu *battle* yang akan ketika diklik akan menampilkan panel pemilihan tingkat kesulitan seperti yang ditunjukan oleh Gambar 3.22. Ketika *player* memilih dari salah satu tingkat kesulitan, maka *game* akan memanggil *scene* untuk memulai pertarungan *boxing* dengan komputer. Setelah itu menu kedua adalah menu *training* yang akan memanggil *training scene* untuk memberikan informasi mengenai gerakan pertahanan dan serangan pada *boxing*. Setelah itu menu ketiga adalah menu yang akan menampilkan panel yang berisi individu yang terkait dalam rancang bangun *game* dan yang terakhir adalah menu ke empat. Menu keempat adalah menu yang menampilkan panel yang memuat 5 *score* terbaik dari hasil pertarungan. Gambar 3.23 menunjukkan *mockup* dari panel *leaderboard*.



Gambar 3.22 *Mockup* Panel *Difficulty*

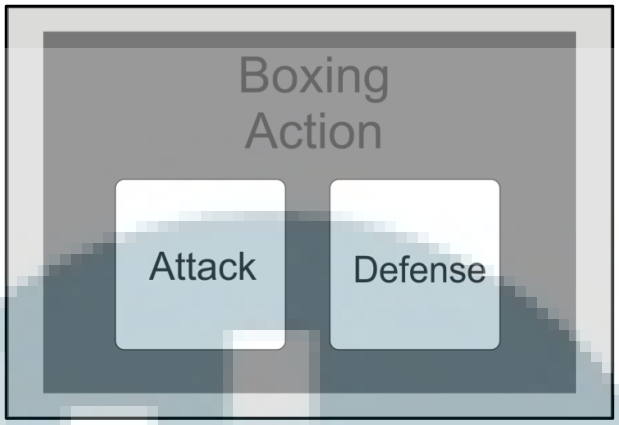


Gambar 3.23 *Mockup* Panel *Leaderboard*



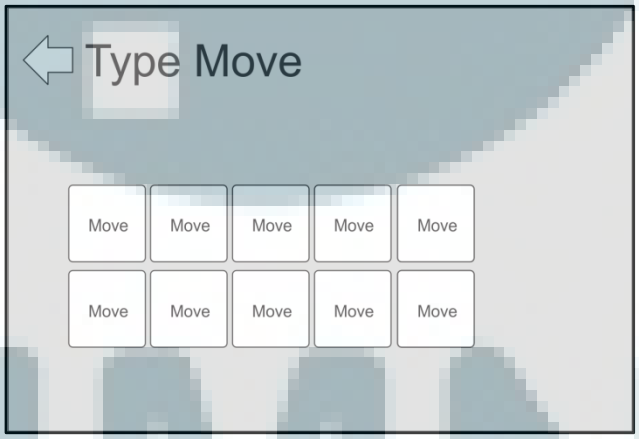
Gambar 3.24 *Mockup Player Interface Loading Screen*

Gambar 3.24 menunjukkan *mockup* untuk tampilan antarmuka *loading scene* yang akan memberikan informasi biografi mengenai petinju-petinju yang terkenal untuk memberikan tambahan informasi kepada pengguna dan *loading scene* ini hanya akan ditampilkan saat perpindahan *scene* terjadi.



Gambar 3.25 *Mockup Player Interface* Pemilihan Tipe *Training*

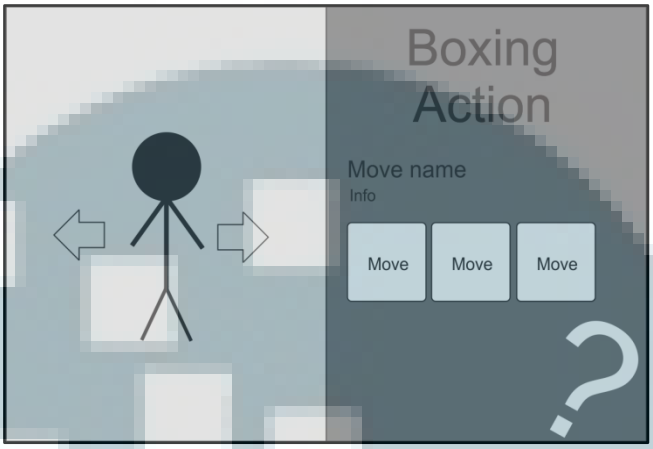
Gambar 3.25 menunjukkan *mockup* untuk tampilan antarmuka sebelum memasuki *scene training*. Jika pengguna memilih menu *attack*, *player* akan diberikan daftar gerakan serangan dan jika memilih menu *defense*, *player* akan diberikan daftar gerakan pertahanan.



Gambar 3.26 *Mockup Player Interface Training Scene*

Gambar 3.26 menunjukkan *mockup* untuk tampilan antarmuka *training* yang berisikan daftar gerakan yang akan ditampilkan sesuai dengan pilihan pengguna di menu sebelumnya. Ketika *player* memilih salah satu gerakan *player*

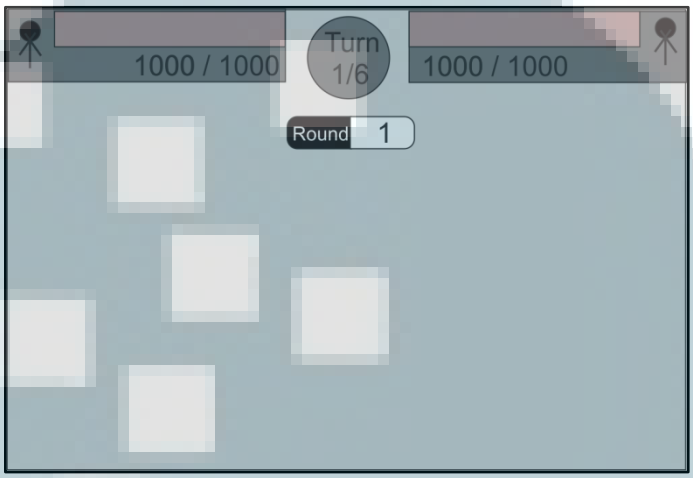
akan ditampilkan *scene* contoh gerakan tersebut dengan menggunakan animasi *boxing* 3D yang dibuat oleh Waroath.



Gambar 3.27 *Mockup Player Interface Training Move Animation and Counter*

Gambar 3.27 menunjukkan *mockup* untuk tampilan antarmuka untuk menampilkan animasi gerakan *boxing*. *player* dapat memutar karakter animasi untuk melihat gerakan karakter dari sudut pandang yang berbeda dengan menekan tombol panah untuk memutar sesuai arah panah atau dengan menggunakan *gesture drag mouse* dengan klik pada karakter lalu geser tangan ke kanan untuk memutar ke kanan dan geser tangan ke kiri untuk memutar ke kiri. *Player* juga ditampilkan daftar gerakan *counter*-nya. Jika *player* memilih untuk melihat daftar serangan di menu sebelumnya, tampilan *move animation* ini akan menampilkan gerakan serangan yang dipilih dan memberikan daftar gerakan untuk menghindari serangan tersebut dan jika *player* memilih untuk melihat daftar pertahanan di menu sebelumnya, *player* akan ditampilkan gerakan pertahanan yang dipilih beserta gerakan serangan yang dapat dihindari dengan gerakan pertahanan tersebut.

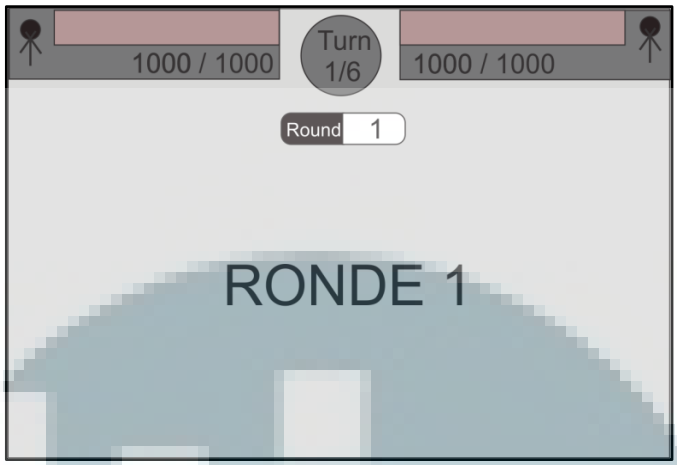
Gambar 3.28 menunjukkan *mockup* untuk tampilan antarmuka pada *battle scene*. Terdapat *health bar player* sebelah kiri dan *health bar enemy* sebelah kanan di bagian atas dan terdapat lingkaran yang menampilkan *turn* dan dibawah lingkaran *turn* terdapat panel yang menampilkan *round*.



Gambar 3.28 *Mockup Player Interface Battle Scene*

Kalimat perintah akan ditampilkan di tengah layar seperti pada Gambar 3.29, Gambar 3.30, dan Gambar 3.31.

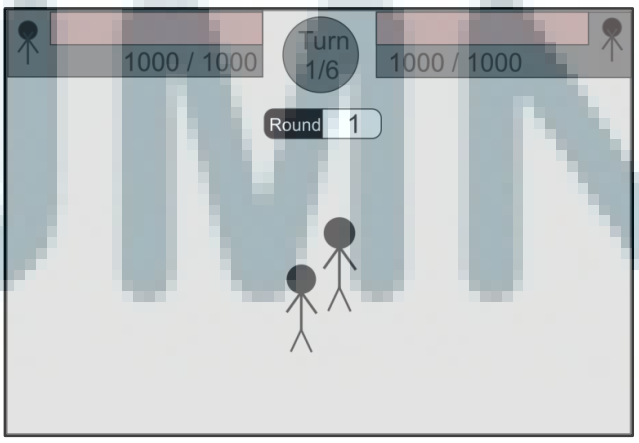
Gambar 3.29 Bersiap Di dalam *Battle Scene*



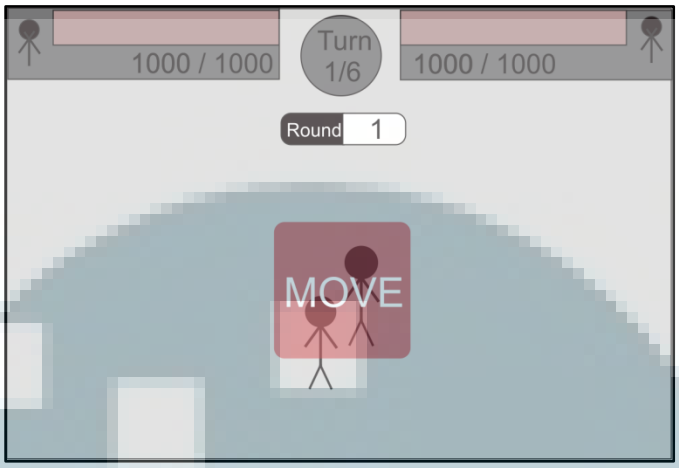
Gambar 3.30 Ronde 1 di dalam *Battle Scene*

Gambar 3.31 Mulai Di Dalam *Battle Scene*

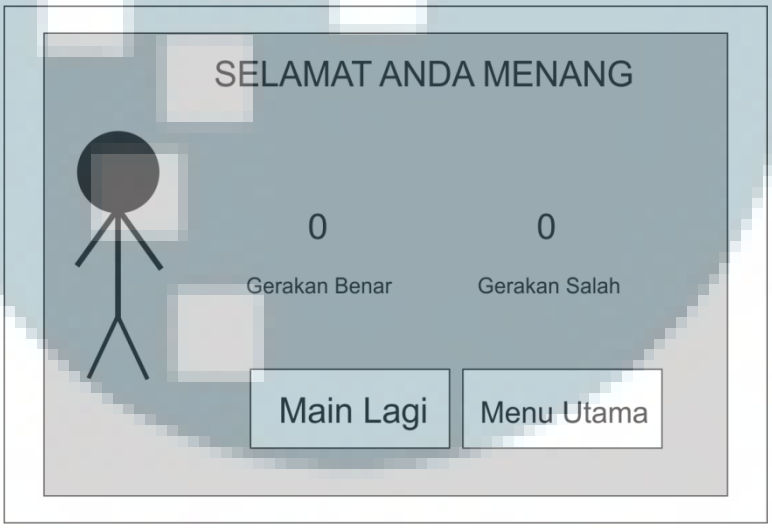
Gambar 3.32 adalah *view* dari kamera utama dan Gambar 3.33 merupakan *mockup* untuk menampilkan gerakan yang telah di-*input* oleh *player*.



Gambar 3.32 *View* Kamera Pada *Battle Scene*



Gambar 3.33 Panel yang Menampilkan Gerakan Pada *Battle Scene*



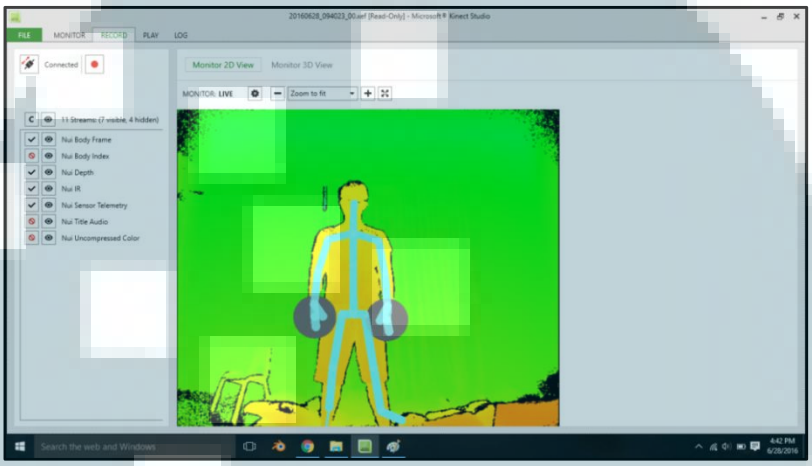
Gambar 3.34 *Mockup Result* Panel *Battle Scene*

Gambar 3.34 menunjukkan *mockup* untuk panel *result* yang akan menampilkan hasil pertarungan dan terdapat *button* main lagi dan *button* menu utama.

3.6 Pembuatan Gesture

Pembuatan *gesture* dibuat dengan menggunakan sensor Kinect versi 2 sebagai *hardware*. Kinect Studio dan Visual Gesture Builder merupakan *software*

akan digunakan untuk membantu proses pembuatan *gesture*. Pembuatan *gesture* dimulai dengan perekaman *skeleton* menggunakan Kinect Studio. Terdapat 3 *tabs* di dalam Kinect Studio dan yang akan dipakai untuk melakukan rekaman *gesture* adalah *tab* kedua yang merupakan *tab record* dengan tampilan seperti pada Gambar 3.35.

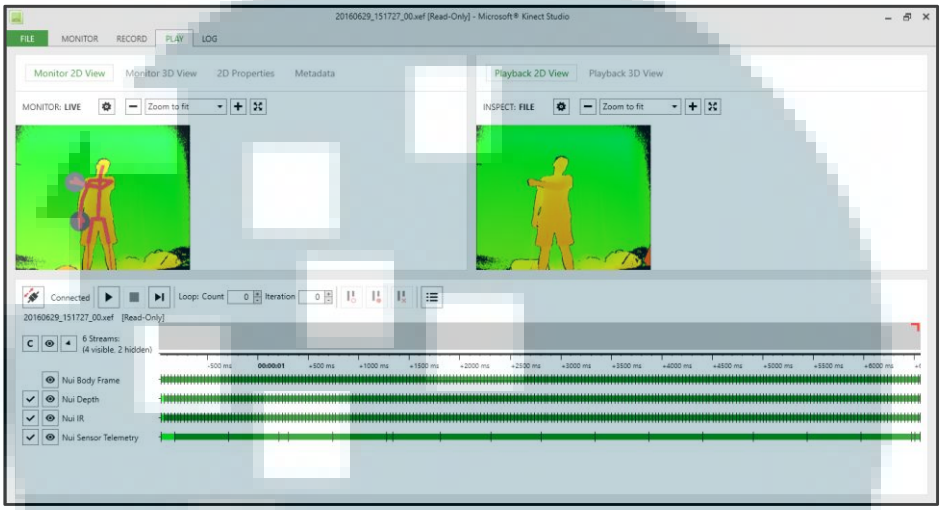


Gambar 3.35 *Mockup Result* Panel *Battle Scene*

Untuk merekam *gesture* dapat klik pada button *record* yang memiliki simbol bulat merah pada Gambar 3.35 dan secara otomatis akan merekam gerakan *skeleton* dan simbol *button* akan berubah menjadi kotak hitam seperti pada Gambar 3.36.

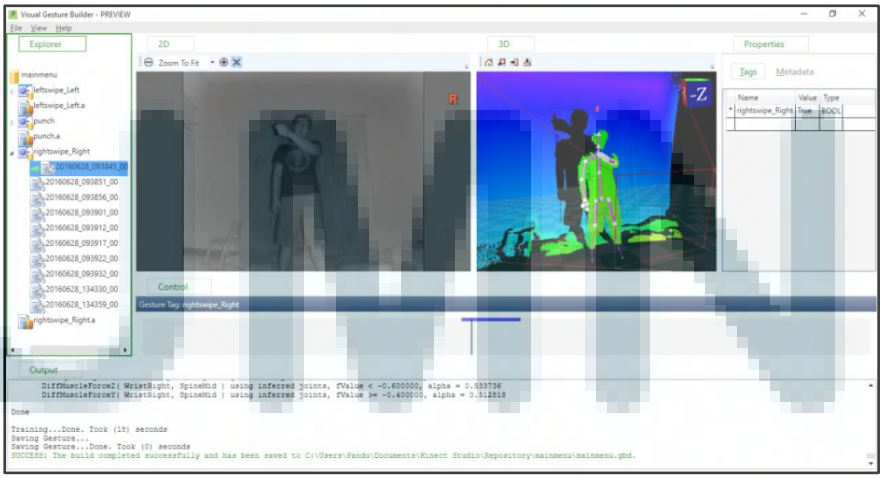
Gambar 3.36 Tampilan *Tab Record* Kinect Studio

Setelah perekaman selesai Kinect Studio akan menyimpan *file*-nya dengan extension .xef (eXtended Event File) ke sebuah *folder*. *File* tersebut akan digunakan sebagai bahan pelatihan *gesture* dan akan menampilkan tampilan seperti pada Gambar 3.37 setelah rekaman selesai.



Gambar 3.37 Tampilan *Tab Record* Saat Merekam Pada Kinect Studio

Proses pelatihan *gesture* dilakukan dengan menggunakan Visual Gesture Builder seperti contoh proses pembuatan *gesture swipe left* untuk *game* pada *mainmenu scene* seperti pada Gambar 3.38.



Gambar 3.38 Tampilan Visual Gesture Builder

Untuk membuat *gesture recognition* yang baik diperlukan lebih dari satu rekaman untuk tiap-tiap *gesture*. Semakin banyak rekaman semakin baik pula *gesture recognition* yang dilakukan. Visual Gesture Builder akan memproses *xef file* dan mempelajarinya dengan AI lalu ketika proses pembelajaran selesai, Visual Gesture Builder akan menghasilkan file dengan extension *.gbd* (*Gesture Builder Database*). *File* inilah yang akan digunakan untuk *data recognition* di *game*.